Copyright

by

Sebastian Goll

2009

The Thesis committee for Sebastian Goll

certifies that this is the approved version of the following thesis:

Design and Implementation of the Disease Control System DiCon

APPROVED BY

SUPERVISING COMMITTEE:

Supervisor: _____

Manfred Fink

_____

Lauren Ancel Meyers

Design and Implementation of the Disease Control System DiCon

by

Sebastian Goll

Thesis

Presented to the Faculty of the Graduate School
of the University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree of

Master of Arts

The University of Texas at Austin
December 2009

To my parents,

for believing in me.

Acknowledgments

This thesis represents some of the results of the research I had the wonderful chance to participate in during the 16 months that I spent as a graduate student at the University of Texas at Austin.

First and foremost, I would like to express my deepest thanks to my advisor Lauren Meyers. Had she not been, I would not have been able to come to Texas in the first place: without knowing or ever meeting me, she gave me the chance to work with her as her graduate research assistant, when the scholarship program awarded by the DAAD (*Deutscher Akademischer Austauschdienst*, German Academic Exchange Service) was suspended for the 2008/2009 season.

I would also like to thank Nedialko Dimitrov. Besides working with me and being my "second advisor" I found in him a great friend who not only introduced me to Austin and the States in general but also has this wonderful ability that motivated me whenever I stopped feeling confident in my work. Without him, this thesis would not have been possible.

My thanks also goes to Manfred Fink who first initiated the exchange program between Würzburg University and the University of Texas at Austin many years ago, and who has been responsible for this program ever since. He was the one that put out all the stops and, within a couple of mere weeks, found ways and means to bring six Würzburg students to Austin without the help of the usual scholarship.

I would also like to thank the other members of Lauren's research group, and especially Thomas Hladish with whom I shared my office: it was always great talking to him about Linux, software, programming, and computers in general. Also, he gave me a wonderful first Thanksgiving.

Last but not least, I would like to express my gratitude to my friends Benjamin Erk and Stefan Bedacht. They not only shared an apartment with me during my stay

in Austin, but are also responsible for sparking in me the idea of going to the States in the first place. Without those guys I would probably never have left Germany and would surely have missed out on this wonderful experience of spending nearly one and a half years at the great University of Texas at Austin. Thank you!

Sebastian Goll — The University of Texas at Austin

December 2009

Design and Implementation of the Disease Control System DiCon

by

Sebastian Goll, M. A.

The University of Texas at Austin, 2009

Supervisor: Manfred Fink

This work describes the design and implementation of the Disease Control System DiCon (pronounced ['daɪkɒn]), providing a general framework for solving optimization problems on distributed computer systems. The central aspects of DiCon are discussed, as are decisions made while realizing the system. Several implementation-specific details are highlighted. Real-world applications show the system's flexibility and demonstrate the potential impact DiCon has on public-health decision making.

Contents

Chapter 1 Introduction **1**
1.1 Definitions . . . 2
1.2 Framework . . . 4
1.2.1 Run-time . . . 5
1.3 Environment . . . 6
1.3.1 Simulators . . . 6
1.3.2 Optimizers . . . 7

Chapter 2 Design **8**
2.1 Goals . . . 8
2.2 Concepts . . . 11
2.2.1 Distributed computing . . . 11
2.2.2 Message Passing Interface (MPI) . . . 12
2.2.3 Example cluster: TACC Lonestar . . . 15
2.3 Communication protocols . . . 15
2.3.1 Node intercommunication . . . 16
2.3.2 Google Protocol Buffers . . . 16
2.3.3 Simulator interface . . . 17
2.3.4 Optimizer interface . . . 18

Chapter 3 Implementation **21**
3.1 Programming language . . . 21
3.2 Node managers . . . 22
3.2.1 Master node . . . 24
3.2.2 Processing node . . . 26

3.3 Job state machine 28
3.4 Specifier parser 30
3.4.1 Specifier grammar 30
3.4.2 Lazy lists and values 37

Chapter 4 Application **42**
4.1 Distribution of antivirals 42
4.1.1 Model 42
4.1.2 Results 44

Chapter 5 Conclusion **50**

Appendix A Configuration **52**
A.1 `[global]` section 52
A.2 `[job-...]` section 57

Appendix B Source code **60**
B.1 Miscellaneous 61
B.2 Main system 64
B.3 Optimizers 211
B.4 Simulators 221

Appendix C GNU General Public License **228**

Bibliography **245**

Vita **250**

Chapter 1

Introduction

The simulation of infectious diseases is quite common in epidemiology, the branch of medical science that deals with the incidence, distribution, and control of disease in a population [1]. With these simulations, researchers can have a look at how a given infectious disease, like the common flu or smallpox, spreads among a human, and sometimes animal, population.

In the past few years, many studies have focused on simulating the spread and spatial dynamics of disease transmission [2–5], with a particular, more recent focus on the influence of airline travel [6, 7]. While other studies investigate possible intervention strategies to mitigate the spread of an epidemic disease [8–14], most studies share one common aspect: the search space for tunable parameters that influence the disease simulation or intervention is rather small.[1]

Often it is rewarding to know how the behavior of some disease changes, or how the outcome of an intervention policy looks different, when disease parameters are varied smoothly from one extreme to the other. This is especially true since most of the disease parameters are known only within certain ranges, particularly when the disease in question has been introduced only recently.

Similarly, for any set of disease parameters there is an optimal intervention strategy within the space defined by all intervention parameter values. Obviously, it makes

[1] Parameters that influence the disease simulation include the basic reproduction number, or the mean infectious period of a disease, while parameters that influence the intervention include the number of available flu shots, or when to distribute antivirals.

a huge difference in how easily a disease can be contained when a different number of vaccine shots are available, but it might also be important how these shots are distributed spatially, or when they are released to the public. Combining the results from many individual simulation runs makes it possible to identify the best intervention policy in each case.

Both scenarios share the following idea: many independent simulations are run, each with different disease and intervention parameters, and the results are recorded. Intervention parameters can be chosen at random, or, when an optimal strategy is sought, depending on some optimization algorithm that takes into account the results from all previous simulation runs with similar disease parameters.

The ready availability of powerful computer clusters today makes it possible to run literally millions of disease simulations within hours, solving for optimal policies, or finding best fits to actual, real-world disease data.

This work presents the general framework and optimization system DiCon that makes it easy for researchers to adapt their existing disease simulators and run them on many different disease and intervention parameters simultaneously, identifying the best intervention policies within a given policy space.

DiCon makes efficient use of distributed and parallel computing, and features a simple, straightforward configuration to specify the actual problem set. DiCon also comes with a number of predefined optimization algorithms, while still allowing users to implement their own algorithms when necessary.

1.1 Definitions

The following terms are used consistently throughout this work.

Policy Set of parameter values that influence a disease simulation. Parameters can be disease parameters or intervention parameters. Typical policies are: number of antiviral doses to distribute among the population each month, list of certain cities to treat specially during disease spread, list of airports to shut down whenever the prevalence in the corresponding city reaches a critical threshold.

Simulator User-defined program that simulates the spread of an infectious disease under a certain policy and returns a reward value that describes how "good" or

"bad" the given policy is. The simulator uses disease and intervention parameters defined either in a static configuration file or given as part of the current policy. Simulations can be stochastic: a simulator is allowed to return a different reward value each time it is run for the same policy.

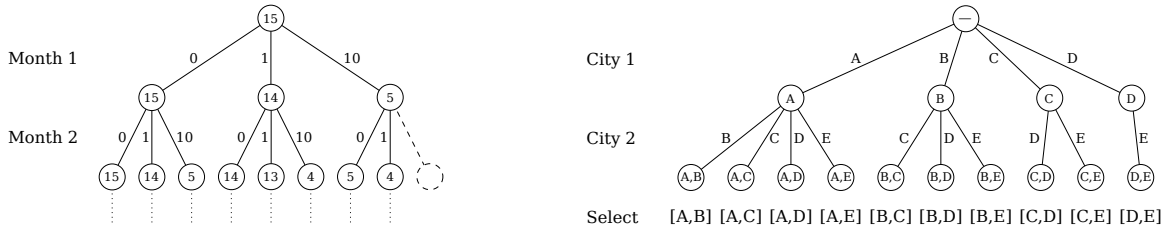
Reward value Real number between 0 and 1 used by the simulator to describe how "good" (1) or "bad" (0) a given policy is. Typical metrics used to determine the reward value of a policy are: fraction of population that remains susceptible throughout the entire disease epidemic; number of cities that do not get the disease, divided by the total number of cities; number of cases prevented through disease intervention strategies, divided by the total number of cases.

Policy space User-defined set of available policies. Each run of the system tries to find the best policies in this set. The best policies are the policies that return the highest average reward values on simulation. Each policy space is defined as a tree, where each policy is equivalent to a list of edges from the root node to a leaf node in the tree. This is done to improve the performance of certain optimization algorithms used to find the best policies.

Optimizer Implementation of an optimization algorithm that selects policies to simulate next, depending on the outcome of all previous simulations. An optimizer is implemented as a shared library and can be user-defined. After each simulation, the optimizer uses the reward value returned by the simulator to update an internal data structure; this structure is used to list the best policies found in the policy space at the end of each optimization job.

Optimization job Entity that describes a single optimization task. A job describes which simulator program to run and optimization algorithm to use, and defines all other relevant parameters, such as command-line arguments to the simulator, additional parameters to the optimizer, and general configuration settings such as when to take snapshots of the optimizer's progress, or where to store log files.

System run Single execution of the DiCon program. Each run is given a directory in the file system. The system reads in a configuration file in that directory which describes at least one optimization job. The system then spawns simulator and



(a) Time-based distribution. (b) Selection of cities.

Figure 1.1: Sample policy trees. Each policy tree describes a policy space; policies are equivalent to the paths from the root to each leaf in the tree. (a) Time-based distribution of antivirals. Each month, the given number of antivirals (in millions) is distributed among the cities of a network, according to some unspecified distribution scheme. The number inside each node is the number of antivirals left in the initial stockpile. An additional constraint on the maximum number of months is required to keep this policy tree finite. (b) Selection of particular cities. A total of 2 out of 5 cities are selected. During the simulation, those two cities are quarantined whenever the fraction of infected individuals inside each city reaches an unspecified threshold. The remaining three cities are not treated in a special way.

optimizer programs on the nodes in the computer cluster as necessary to process all declared jobs in sequence. If a run is interrupted, another run can be created using the same directory: it will automatically resume all unfinished jobs.

Though DiCon is the Disease Control System, it is not required at any point that the simulations described are actual disease simulations. The framework is general enough to be run with any simulation that depends on some variable parameters, and it can therefore be used to solve optimization problems coming from any background.

1.2 Framework

The DiCon framework consists of the following modules.

Manager Reads in configuration of jobs; spawns optimizer and simulator programs.

Simulator User-provided; defines policy space; runs simulation with a given policy.

Optimizer Collects results from simulations; selects which policies to simulate next.

Both simulator and optimizer programs can be user-defined, though DiCon ships with a number of different, predefined general-purpose optimizers. In order to help more

advanced optimization algorithms in making decisions, the policy space is represented in the form of a tree. In fact, for many types of disease intervention policies it makes perfect sense to use a hierarchical, tree-like structure; some examples for this are shown in figure 1.1 on the preceding page.[2]

Apart from the tree property, policies are entirely defined by the user's simulator program, and there is no limitation on what they may represent—in fact, as the user's own simulator is the only part of the system that will ever "see" each element in a policy path again, edges in the policy tree need not even follow a special format: they can be arbitrary strings of data, such as serialized representations of a more complex, private data structure.

1.2.1 Run-time

DiCon is run using the specific Message Passing Interface (MPI) wrapper provided by the computer cluster in question, such as `mpirun`. This wrapper program is responsible for starting separate instances of DiCon on every computer in the cluster and establishes the communication channels between these instances. The MPI standard and its use in DiCon is explained in more detail in section 2.2.2 on page 12.

The DiCon executable requires only a single command-line argument: the name of a directory in the file system. On invocation, DiCon changes the current working directory of the process to the directory given as command-line argument; this directory is then used as working directory for all child processes that are spawned by the system, in particular, it is used as working directory for all user-defined simulator programs that are run by DiCon.

Initially, the working directory contains only a single configuration file that describes the optimization jobs to process in sequence. As DiCon is running, additional files are put into this directory, such as log files and, more importantly, result files.

When checkpointing is enabled for an optimization job, the checkpoint data is also put into the working directory. Since DiCon recognizes checkpoint files when starting an optimization job, resuming interrupted system runs is made particularly

[2]Though each policy space should be represented as a tree, this is not a restriction as such: a simulator program might very well opt for representing its policy space as a degenerate tree where each node attached to the root is a leaf. The policy tree then becomes a simple, flat list of policies.

easy: the user simply specifies the same working directory when restarting DiCon, and the system will automatically take care of skipping jobs that have already been completed, and resuming jobs that have been interrupted.

When DiCon is run, the MPI implementation makes sure that an instance of the DiCon executable is started on each node in the computer cluster that is to be used by the system. All nodes except one are used as processing nodes that assume either an optimizer or simulator role as the system is running, the remaining node becomes the master node which is responsible for managing the processing nodes and distributing jobs among them.

1.3 Environment

DiCon is written in C++ [15] and makes heavy use of the Boost C++ Libraries [16]; it also uses Google's Protocol Buffers [17], and the GNU Multiple Precision Arithmetic Library, or GMP [18]. It runs on distributed computer systems that provide an implementation of the Message Passing Interface, or MPI [19].

The following compilers and libraries are known to work with DiCon: GNU Compiler Collection 4.3.4, Boost C++ Libraries 1.40.0, Google's Protocol Buffers 2.2.0, GNU Multiple Precision Arithmetic Library 4.3.1, Open MPI 1.3.3.

1.3.1 Simulators

Simulator programs can be arbitrary computer programs provided by the user. They communicate with the rest of the system through standard input and output using Google's Protocol Buffers and are spawned by the system as necessary. Three types of messages are defined: `children` gets the list of outgoing edges at any given node in the policy tree, `simulate` runs a simulation with the given policy, and `display` transforms a policy into a human-readable representation that is used in the final output. The exact protocol is discussed in more detail in section 2.3.3 on page 17.

Each simulator program is kept running as long as possible. This implies that any initialization that might be required needs to be done only once, saving time when doing subsequent simulations. If, however, an existing simulator is written in a non-reentrant way and cannot be easily run with another policy, the simulator program

may exit at any time; it is automatically restarted by DiCon as necessary.

Since simulator programs communicate with the rest of the system only through standard input and output, there is no constraint on the programming language used: it can be any language that has an implementation of Google's Protocol Buffers. While Google provides official implementations only for C++, Java, and Python, as of 2009, many other languages, such as C#, Common Lisp, D, Erlang, Haskell, Perl, PHP, Ruby, or Visual Basic, are supported through third-party add-ons. [20]

Simulator programs can be started with arbitrary command-line arguments, specified in DiCon's global configuration file. Any output the simulator program sends to standard error is redirected to a log file; this is useful when diagnosing errors, or when getting debug information or run-time statistics out of the simulator.

1.3.2 Optimizers

Optimization algorithms are implemented in the form of shared libraries; communication with the rest of the system is accomplished through C library calls. The interface is slightly more complex than the one used for simulator programs and requires a total of six messages.

The optimizer messages do the following: `initialize` sets up the optimizer and is called before any other message, `get_policy` asks for another policy to simulate next, `update` sets the policy's reward value returned from a simulation, `policies` compiles a list of best policies found so far, `dump_state` writes the optimizer's internal state to a file, and `get_error` returns an error message after an error occurred. The exact interface is discussed in more detail in section 2.3.4 on page 18.

Each optimizer library is required to implement the ability to dump its internal state to a file from which it can later be resumed. This requirement makes it possible to rerun DiCon jobs after they have been interrupted.

Resuming DiCon runs comes in handy in situations where the policy of a computer cluster or supercomputer requires all jobs to finish within a certain amount of time, such as within 48 hours. With optimization algorithms being resumable, any job can simply be resubmitted: DiCon takes care of continuing from the most recent state automatically.

Chapter 2

Design

The following sections describe the general design of DiCon, trying to do so without delving too much into implementation-specific details. While an occasional glance at implementation-specific design questions cannot be avoided, the main focus lies on the important aspects that influenced the design of DiCon as it is now.

2.1 Goals

When first coming up with the idea of an optimization framework, several aspects had to be considered. As the intent of such a framework is to become widespread in use among many research groups, it must be intuitive and easy to use: a lot of effort is required to make oneself familiar with a new system, and not many people find it worthwhile to do so when the entire concept is unnecessarily complex or designed in an overtly counterintuitive way.

Despite being easy to use, a framework aiming to be generally applicable must also be flexible enough to address many different problems. Thus, it must not only be customizable to be fed with the specific simulation in question, but also allow for the exchange of the optimization algorithm to be used: while DiCon ships with several predefined algorithms, new algorithms are discovered regularly; also, there might be highly specialized, handcrafted algorithms particularly suited for the problem in question that, for one reason or another, cannot be included in the official DiCon distribution.

It should be possible to change a lot of the system's internal parameters: How

does the system distribute jobs among the processing nodes in the computer cluster? Where are log files kept and which types of message end up in which log file? How much time is the system allowed to spend on each job? Depending on the environment used it might be necessary to twist any of these knobs. The framework must provide means to do so; where this is not required, though, it must fall back to meaningful default settings.

In addition to all that, the system must still perform well. A general framework is not of much use when it comes at a great cost of resources, be it run-time or memory usage. Since the fundamental concept of distributed computing inherently involves a certain level of management and communication between the nodes in the computer cluster, a small overhead cannot be avoided; still, this overhead must be kept as small as possible, minimizing any loss of resources.

Last but not least, the design and implementation of the framework should be simple and concise, and easy to understand and follow, making both future development, and custom modifications and additions as easy as possible. This also makes finding and fixing bugs less challenging.

The goals involved in the design and development of DiCon can thus be summarized as follows.

Ease of use It must be easy enough to manage the system; preferably through a single point of configuration such as a single configuration file. Getting started with DiCon must not involve reading tons of documentation; rather, setting everything up should be intuitive and involve only mildly complex tasks.

Flexibility DiCon must be flexible enough to not only allow for specifying custom simulators but also custom optimization algorithms. Many of the internal parameters must be tunable, yet sensible default settings should be available. The interface between DiCon and both simulator and optimizer programs should be simple and easy to understand.

Performance The overhead of using DiCon over a handcrafted distributed computing environment must be negligible. Also, the overhead over a non-distributed, classical optimization approach should be as small as possible.

Simplicity DiCon should be designed and implemented in such a way as to make

future changes as well as custom modifications to the system itself as easy and straightforward as possible.

These goals have been met as follows.

The entire configuration of any single DiCon run is managed in one single configuration file; in fact, each DiCon instance is started in its own directory which, at first, contains only the main configuration file of the specific run. This file describes the set of optimization jobs to perform, which simulator and optimizer programs to use, and also contains settings for any of the system's internal parameters as required. While running, DiCon fills in the working directory with results as each optimization job completes; also, checkpoints are taken at user-definable intervals, making it possible to resume each job in case it is interrupted.

Both simulator and optimizer programs communicate with the rest of the system by the means of simple communication protocols. These protocols are described in detail in section 2.3.3 on page 17, and section 2.3.4 on page 18, respectively. Simulator and optimizer programs can be written in any programming language, which makes it particularly easy to adapt existing simulations to be used with DiCon. Small wrapper libraries for C++ and Python are provided to make it even easier to implement the required interfaces in these programming languages.

Run-time measurements have shown that the entire raw overhead in a typical optimization run is less than 10 % over the theoretical limit imposed by the corresponding simulation. For this, a simple policy space was provided and each simulator did nothing but wait for one second. Running this job in parallel on as many as 512 processing nodes in the computer cluster *Lonestar* (section 2.2.3 on page 15), and using an exhaustive search algorithm simulating each policy exactly once, took a wall clock time of less than $1.1 \times MN^{-1}$ seconds, where N was the number of processing nodes, and M was the number of policies in the given policy space (≈ 1 million).

The design of DiCon is particularly simple and as straightforward and intuitive as possible. In particular, each node in the computer cluster runs a single-threaded node manager; concurrent programming is not used at all in the implementation of DiCon. Instead, in order to still reach the high performance desirable, non-blocking communication is used. The inherent increase in complexity that comes with non-blocking operation is met by making clever use of inheritance in the object oriented programming model, and establishing several conditions on how non-blocking calls

are processed. The details of this are described in section 2.3.1 on page 16.

2.2 Concepts

The next subsections introduce some fundamental concepts relevant to DiCon, or any distributed optimization system in principal. This includes an introduction to distributed computing in general (section 2.2.1), the Message Passing Interface used in DiCon (section 2.2.2 on the next page), and an exemplary description of the computer cluster *Lonestar* at the Texas Advanced Computing Center (section 2.2.3 on page 15).

2.2.1 Distributed computing

The term *distributed computing* is used for any computer system that, in contrast to a single computer, runs a calculation on more than one machine. A more formal definition might be given as follows. [21]

- The system consists of more than one sequential process.
- Processes communicate with each other using messages.
- Processes have disjoint address space and memory.
- Processes must interact to meet a common goal.

While distributed computing in principle is a very general concept, including by definition applications such as telephone networks, aircraft control systems, the Internet, and the World Wide Web, the remainder of this thesis will focus on the specific application of *parallel computing* in the form of *computer clusters* or modern *supercomputers*, where multiple computers work closely together on the same task, connected to each other typically through fast local area networks.

In the scope of this work, the set of machines or computers that are working together is thus referred to as a computer *cluster*, while each individual computer is referred to as a *node* in the cluster. In clusters where each machine has more than one central processing unit (CPU) or CPU core, each CPU or core might be addressed as individual node instead.

History

The concept of distributed computing—concurrent processes communicating by exchanging messages—has its root in operating system architectures studied in the 1960s. [22] The first widespread distributed systems were local-area networks such as Ethernet which was invented in the 1970s. [22, 23] The introduction of ARPANET, the predecessor of today's Internet, in the late 1960s, and the invention of e-mail in the early 1970s, is probably the earliest example of a large-scale distributed application.

The study of distributed computing soon became its own branch of computer science in the late 1970s and early 1980s. Together with this came the fundamentals of parallel computing, such as consistency models and process calculi. [24]

In recent years, the combination of traditional supercomputing with distributed and parallel computing has led to the creation of massive parallel processing systems with thousands of "ordinary" CPUs, in contrast to the highly specialized, custom-designed CPUs used in earlier supercomputers. These parallel systems establish the majority of today's highly-tuned computing clusters. [25]

As of 2009, there is an innumerable amount of computer clusters and other supercomputers available for highly calculation-intensive tasks worldwide. Since 1993, the TOP500 project ranks the fastest supercomputers and publishes an updated list twice a year, aiming to provide a reliable basis for tracking and detecting trends in high-performance computing. [25] Part of the November 2009 list, released November 17, 2009, at the SC09 Conference, is shown in table 2.1 on the next page.

2.2.2 Message Passing Interface (MPI)

With more supercomputers becoming available worldwide came a great number of different system libraries. To make it easier to run an application not only on a single supercomputer but also on others, and still make use of the distributed and parallel computing paradigms, the MPI Forum started in 1992 an activity that would lead to the creation of the *Message Passing Interface*, or short, MPI.

At that time, proprietary message passing libraries were available on several parallel computer systems, and were used to develop significant parallel applications. These libraries, though, were not compatible with each other, meaning that code developed on one system could not easily be ported to another. [26]

Rank	Site	Computer/Vendor/Year	Cores	$R_{\max}$	R_{peak}	Power
1	Oak Ridge National Laboratory (United States)	"Jaguar" (Cray Inc. 2009) Cray XT5-HE Opteron Six Core 2.6 GHz	224,162	1759.00	2331.00	6950.60
2	DOE/NNSA/LANL (United States)	"Roadrunner" (IBM 2009) PowerXCell 8i 3.2 GHz/Opteron DC 1.8 GHz	122,400	1042.00	1375.78	2345.50
3	NICS/Univ. of Tennessee (United States)	"Kraken XT5" (Cray Inc. 2009) Cray XT5-HE Opteron Six Core 2.6 GHz	98,928	831.70	1028.85	—
4	Forschungszentrum Jülich (FZJ) (Germany)	"JUGENE" (IBM 2009) Blue Gene/P Solution	294,912	825.50	1002.70	2268.00
5	SuperComputer Center/NUDT (China)	"Tianhe-1" (NUDT 2009) Xeon E5540/E5450/ATI Radeon HD 4870 2	71,680	563.10	1206.19	—
6	NASA/ARC/NAS (United States)	"Pleiades" (SGI 2009) Xeon QC 3.0 GHz/Nehalem EP 2.93 GHz	56,320	544.30	673.26	2348.00
7	DOE/NNSA/LLNL (United States)	"BlueGene/L" (IBM 2007) eServer Blue Gene Solution	212,992	478.20	596.38	2329.60
8	Argonne National Laboratory (United States)	— (IBM 2007) Blue Gene/P Solution	163,840	458.61	557.06	1260.00
9	TACC/Univ. of Texas (United States)	"Ranger" (Sun Microsystems 2008) SunBlade x6420, Opteron QC 2.3 GHz	62,976	433.20	579.38	2000.00
10	SNL/NREL (United States)	"Red Sky" (Sun Microsystems 2009) Sun Blade x6275, Xeon X55xx 2.93 GHz	41,616	423.90	487.74	—
⋮	⋮	⋮	⋮	⋮	⋮	⋮
105	TACC/Univ. of Texas (United States)	"Lonestar" (Dell 2007) PowerEdge 1955, 2.66 GHz	5,848	46.73	62.22	—
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 2.1: Some of the supercomputers on the TOP500 list as of November 2009. [25] $R_{\max}$ and R_{peak} are the maximal performance achieved in the LINPACK test, and the theoretical peak performance of the system, respectively, and are given in TFLOPS. Power, where available, is for the entire system and is given in KW.

When several public-domain efforts independently demonstrated that a message passing system could be implemented both in an efficient and portable way, the time was ripe to define both the syntax and semantics of a standard core of library routines that would be useful to a wide range of users, and efficiently implementable on a wide range of computers. This effort was undertaken in 1992 by the MPI Forum, a group of more than 80 people from 40 organizations, representing vendors of parallel systems, industrial users, industrial and national research laboratories, and universities. [26]

In May 1994, version 1 of the MPI standard was released. This first revision defined the user interface and functionality for a wide range of message-passing capabilities while being as portable across different machines as the programming languages typically used in parallel computing: Fortran, C, and C++. This meant that the same source code could now be compiled and run on a variety of different machines as long as an MPI library was available. [26]

Since 1994, several revisions and additions to the original standard have been released. The current versions, as of 2009, are MPI-1 1.2 and MPI-2 2.0, and developments for a possible MPI-3 standard are on their way.

Use in DiCon

Despite the huge framework defined by the MPI, with hundreds of specialized library calls, DiCon makes use only of the most fundamental point-to-point communication—in this way, DiCon still takes advantage of the portability of the MPI specification, allowing it to run on virtually any of today's distributed systems, without using the collective functions also defined by the MPI.

The reason for this lies in the nature of the operations performed within DiCon: depending on both the actual policy space and optimization algorithm used for each optimizer job, each simulation might take an unpredictable amount of time. Using the more advanced collective functions offered by the MPI would introduce additional idle time in the system in cases where simulations do not finish within approximately the same time, since the reduction of results would have to wait on all involved simulators.

Calls to the MPI library are not made directly; instead, the Boost MPI wrapper library is used. This makes it more intuitive to use the MPI in C++, through the use of object-oriented encapsulation, and also takes care of the low-level aspects required by the MPI, such as memory management, and serialization of data to send and

receive. In so doing, it also avoids bugs introduced by misusing the MPI library.

2.2.3 Example cluster: TACC Lonestar

DiCon was developed and tested on the Dell Linux Cluster *Lonestar* at the Texas Advanced Computing Center (TACC) at the University of Texas at Austin. As of November 2009, *Lonestar* ranks number 105 on the international TOP500 list. [25]

Lonestar contains 5,840 cores within 1,460 Dell PowerEdge 1955 compute nodes, 16 PowerEdge 1850 compute-I/O server-nodes, and 2 PowerEdge 2950 (2.66 GHz) login and management nodes. Each compute node has 8 GiB of memory, each of the two login/development nodes has 16 GiB. The system storage includes a 103 TiB parallel Lustre file system used for work data, and 106.5 TiB of local compute-node disk space (73 GiB per node). An InfiniBand switch fabric, employing PCI Express interfaces, interconnects the nodes (I/O and compute) through a fat-tree topology, with a point-to-point bandwidth of 1 Gbit/sec (unidirectional speed). [27]

Compute nodes have two processors, each a Xeon 5100 series 2.66 GHz dual-core processor with a 4 MiB unified (Smart) L2 cache. Peak performance for the four cores is 42.6 GFLOPS. Some of the key features of the Core micro-architecture are: dual-core, L1 Instruction cache, 14 unit pipeline, eight prefetch units, Macro Ops Fusion, double-speed integer units, Advanced Smart (sharing) L2 cache, and 16 new SSE3 instructions. The memory system uses Fully Buffered DIMMs (FB-DIMMs) and a 1333 MHz (10.7 Gbit/sec) front side bus. [27]

2.3 Communication protocols

DiCon makes use of several different message-oriented protocols to implement communication between the individual parts of the system.

Since DiCon uses all except one node in the computer cluster as processing nodes, and the remaining node as the master node that manages all other nodes, messages have to be passed from the master node to each processing node and back.

Processing nodes spawn simulator programs and call optimizer libraries, each with a different interface, so messages have to be defined for these types of communication as well.

2.3.1 Node intercommunication

Each processing node's sole responsibility is to execute the requests it receives from the master node in the cluster. It does so in a blocking fashion by waiting until each request is finished, before sending out an answer back to the master node to indicate so. Each processing node executes only one request at a time, and requests are processed in the order they are received. Processing nodes operate independently from one another since there is no communication between two processing nodes.

The master node works asynchronously, with regard to both sending and handling requests: the function call for sending requests returns immediately, and answers are only handled when explicitly requested by calling a special function that waits for the first answer to arrive. This way multiple requests can be pending, and processing nodes that finish earlier than others can do so without having to idle.

After doing the initial start-up of the system, the master node waits for the first processing node to finish its request. Only then is the system's internal state updated and new requests are sent out. The job state machine responsible for this mechanism is explained in more detail in section 3.3 on page 28.

Due to the nature of MPI, when the master node sends out multiple requests to a single processing node, requests will arrive in the order they are sent. Since requests are executed in order, the answers sent back to the master node will also arrive in this order. This, in turn, leads to each answer being handled only when all previous requests to this node have finished. It is this invariance that makes implementing the internal state machine feasible in the first place.

Messages are sent between the master node and processing nodes using the Boost MPI wrapper library. This library takes care of all the necessary encapsulation and serialization of data. Since all nodes in the computer cluster are running on the same hardware and computer system, this encapsulation and serialization can be done in an efficient, and not necessarily portable, way.

2.3.2 Google Protocol Buffers

Google's Protocol Buffers is a free library offering both language-neutral and platform-independent encoding of structured data in an efficient yet extensible format. [17] Google itself uses Protocol Buffers for almost all of its internal Remote Procedure

Call (RPC) protocols and file formats. [17]

Official implementations of the Protocol Buffers library are offered by Google for the programming languages Java, C++, and Python, while other languages are supported through third-party add-ons. [20]

Instead of being a complete RPC framework, Google's Protocol Buffers specifies only the encoding of data into self-contained data packets, being much smaller, faster, and simpler than most comparable data structures often used for similar purposes, such as the Extensible Markup Language (XML).

In DiCon, Protocol Buffers is used for the communication between simulator programs and the rest of the system: DiCon writes packets in the Protocol Buffers format to the standard input stream of each simulator; the simulator decodes those packets and formulates a response, also using Protocol Buffers, which it then sends back to DiCon using its standard output stream.

The Protocol Buffers library was chosen for that purpose since it offers a very clean and simple interface and is available for a multitude of different programming languages, making it feasible to use DiCon with many existing simulators. Since the usage of standard input and output is an inherent part of virtually all programming languages, in contrast to alternative mechanisms such as network protocols or socket access, the protocol itself can be implemented in only a couple of lines of source code.

2.3.3 Simulator interface

The simulator programs doing the actual simulation communicate with DiCon using the following *simulator interface*. This interface is kept very simple in order to make implementing it in different programming languages as easy as possible. In addition to doing simulations, user-defined simulator programs also declare the space available to pick policies from during optimization.

After the initial setup, a total of three methods are defined for DiCon to call on each simulator. In the following notation, arguments of a method are given to the left of the arrow ($\rightarrow$), results are to the right. The notation $[\cdot]$ signifies lists of the given type. **policy_element** can be an arbitrary type, **policy** is a list of **policy_element**, **double** is a real number, and **string** is a string of characters.

`children` : [**policy_element**] → [**policy_element**] The `children` method takes a list of policy elements, representing a path from the root node to a specific node in the policy tree, and returns a list of outgoing edges at this node. When the list of outgoing edges is empty, the node indicated by the parameter is a leaf node; the list of policy elements given as parameter to `children` is then equivalent to a policy.

`simulate` : **policy** → **double** The `simulate` method takes a policy (equivalent to a list of policy elements representing a path from the root node to a leaf node in the policy tree), runs a single simulation with this policy, and returns the resulting reward value. Since simulations are allowed to be stochastic, a different reward value may be returned when running the simulation again with the same policy.

`display` : **policy** → **string** The `display` method takes a policy (equivalent to a list of policy elements representing a path from the root node to a leaf node in the policy tree), and returns a textual, human-readable representation of this policy. It is up to the programmer of the simulator to decide what is a suitable representation.

2.3.4 Optimizer interface

Optimization algorithms communicate with DiCon by using the following *optimizer interface*. Each optimization algorithm is implemented in the form of a shared library with each method defined below being equivalent to an exported C library call.

Instead of directly using the policies as they are returned by the simulator's `children` method, DiCon uniquely maps each serialized policy string to an integer. This integer is then used in place of the real policy whenever one of the optimization algorithm's methods is called—in the same way, policies returned from the optimizer are converted back to the original serialized policy string before passing them on to the simulator's `simulate` method.

Not using the actual policy but an ID instead is done in order to make the implementation of optimizers more memory efficient: since many optimization algorithms need to keep track of the policies they return, storing the serialized policy string in memory directly would imply a waste of resources—policies are paths in the policy tree, and storing many different policies is likely to store the elements at the beginning of each path over and over again. Reducing the memory footprint of these policy

elements significantly reduces the total memory usage inside each optimizer.

For the following description of the methods defined for the optimizer interface, the same notation is used as given in the description of the simulator interface on page 17. In addition to that, multiple parameters to a method are expressed through the Cartesian product $(\cdot \times \cdot)$, whereas the absence of an argument or return value is indicated by $\emptyset$, similar to the `void` type in C-style languages; optional parameters are given as $\langle\cdot\rangle$. The extra type **integer** signifies an integral value, and **policy** is now equivalent to a list of native integer values on the platform used to compile DiCon, as given by the `int` type in C-style languages.

`initialize` : $[\textbf{string}] \times \langle\textbf{string}\rangle \to \emptyset$ The `initialize` method takes a list of custom parameter values defined in the corresponding optimization job, and, optionally, the name of a file previously dumped with `dump_state`; if this parameter is given, the optimizer restores its internal state from this file. The `initialize` method is called exactly once, and before any of the other methods is called.

`get_policy` : $\emptyset \to \langle\textbf{policy}\rangle$ The `get_policy` method takes no argument, but returns a policy to simulate next, or an empty return value indicating that no next policy is available. The optimizer is given access to a special callback function that connects to the simulator's `children` method—performing the internal mapping between serialized policy strings and integer IDs as necessary. Using this callback, optimization algorithms are able to explore the parts of the policy tree they are interested in.

Policies returned by `get_policy` are called *pending*, and `update` must be called exactly once for each pending policy. If `get_policy` returns no next policy, it must continue to do so until `update` is called at least once. If `get_policy` returns no next policy and no policies are pending, the end of the optimization job has been reached.

`update` : $\textbf{policy} \times \textbf{double} \to \emptyset$ The `update` method takes a pending policy, as defined previously at `get_policy`, and the reward value obtained from simulating with this policy. It does not return a value, but updates the optimizer's internal state as necessary.

`policies` : **integer** $\rightarrow$ [(**policy**, **double**)] The `policies` method takes an integral non-negative value, N, and returns up to N best policies, together with their respective average reward values. Best policies are policies whose average reward value from previous calls to `update` is greater than or equal to the average reward values of all other policies. This method is called only when no policies are pending.

`dump_state` : **string** $\rightarrow \emptyset$ The `dump_state` method takes the name of a file, and writes the optimizer's internal state to this file, in some optimizer-dependent, unspecified format, so that it can be completely restored when the optimizer is initialized again at a later time. This method is called only when no policies are pending.

`get_error` : $\emptyset \rightarrow$ **string** The `get_error` method returns a textual, human-readable description of the most recent error. It is called only immediately after one of the other methods indicated an error situation.

Chapter 3

Implementation

In the following sections, some of the more interesting details of the DiCon implementation are highlighted. Since this cannot be a thorough inspection of every aspect of the program, the entire source code of DiCon is also printed in appendix B on page 60.

The implementation of DiCon in general has been done in the most straightforward way, following the principle of least astonishment, thus making code management easy while also greatly reducing the risk of bugs in the source code.

A particular focus has been put on exception-safety and sensible error handling throughout all parts of the program, as well as on the strict avoidance of memory leaks through the use of smart pointer structures and the Resource Acquisition Is Initialization (RAII) design pattern [28].

3.1 Programming language

The main part of DiCon is written in C++ [15], with only minor parts of the interface also provided in Python [29]. DiCon makes heavy use of many of the Boost C++ Libraries [16], such as Assign, Bimap, Conversion, Exception, Filesystem, Foreach, Format, Function, Lambda, Lexical Cast, MPI, Optional, Regex, Serialization, Smart Ptr, Spirit, System, Tuple, and Variant. DiCon also requires the Google Protocol Buffers [17], as well as the GNU Multiple Precision Arithmetic Library, or GMP [18].

In order to operate in computer clusters and other distributed systems, DiCon, through the use of Boost MPI, employs any implementation of the Message Passing Interface (MPI) that is provided by the underlying computer system.

3.2 Node managers

As hinted at in section 2.3 on page 15, DiCon uses all available nodes in the computer cluster as follows: when the system is initialized, a single node is selected to be the master node, while all remaining nodes become processing nodes. The node chosen to be the master node is simply the first node, having node rank zero, within the global MPI "world" communicator.

Communication between nodes in the system follows a star topology with the master node as a central hub. In fact, the master node is the only node that initializes communication requests: processing nodes are expected to execute the requests they receive from the master node, and indicate only whether the request succeeded or failed, together with any possible return values. The design of the message interface used for this node intercommunication is outlined in section 2.3.1 on page 16.

The master node manager as well as the processing node manager are derived from the common base class `NodeManager`. The interface to this class is shown in figure 3.1 on the next page. The `NodeManager` base class provides functionality common to each node in the system, such as access to the actual MPI context, the ability to log various node statistics, and the range of tag values available for node intercommunication.

From this class the two abstract classes `MainNodeManager` and `ProcNodeManager` are derived. Among other methods, these classes define abstract virtual methods[1] that are implemented in the classes `MainNodeManagerImpl` and `ProcNodeManagerImpl`. These virtual methods are used for the actual processing of requests in the processing node manager, and for handling the answers to these requests in the main node manager.

In the `main` function of each node's instance of the DiCon executable, a concrete object of `NodeManager` is obtained by calling the static method `create` which will return either an instance of `MainNodeManagerImpl` or `ProcNodeManagerImpl`, depending on whether the node in question is to be the master node or a processing node in the cluster. Calling the `run` method starts processing inside the actual node manager.

[1] Abstract virtual methods are indicated in C++ by the notation "`=0`" at the end of the method declaration.

```
1  /// Node manager interface.
2  class NodeManager
3    : boost::noncopyable
4  {
5  public:
6    /// Run node manager.
7    virtual void run()=0;
8
9    virtual ~NodeManager() {}
10
11   /// Create node manager.
12   static boost::shared_ptr<NodeManager>
13   create( boost::mpi::communicator &world );
14
15 protected:
16   /// Statistics helper.
17   class StatsEntry
18     : public ::StatsEntry
19   {
20   public:
21     /// Push node onto tree.
22     StatsEntry( NodeManager &manager, const std::string &what, int priority = 0,
23                 const std::string &idle = std::string(), int idle_priority = 0 );
24   };
25
26 protected:
27   /// Create node manager.
28   NodeManager( boost::mpi::communicator &world );
29
30   /// Get MPI communicator.
31   const boost::mpi::communicator &world() const;
32   /// Get MPI communicator.
33   boost::mpi::communicator &world();
34
35   /// Print statistics.
36   void log_statistics();
37
38   /// Get minimum tag value.
39   static int min_tag();
40   /// Get maximum tag value.
41   static int max_tag();
42   /// Get number of tag values.
43   static unsigned tag_count();
44   /// Get next tag value.
45   static int &next_tag( int &tag );
46 };
```

Figure 3.1: Interface to the `NodeManager` class.

Messages are passed between the main node manager and each processing node manager using the Boost MPI library. This library serializes structures that contain as members the parameters and return values, respectively, of the request in question; it also takes care of the necessary memory management that is required by the underlying MPI library. Since all nodes in the computer cluster run the same hardware and operating system, the serialization and encapsulation needs not to be done in a portable way, and can therefore be implemented very efficiently.

3.2.1 Master node

The `MainNodeManager` class defines two kinds of methods: concrete methods that implement the sending of requests to processing nodes, and virtual methods that will be called when an answer has been received from a processing node. The interface to the `MainNodeManager` class is shown in figure 3.2 on the following page.

Each of the concrete methods that send a message to a processing node returns immediately: these methods "return" `void` instead of the actual return value. Instead, whenever the `process` method is called, the main node manager waits for the next answer to any pending request to arrive; the `process` method then, in turn, calls the implementation of the corresponding virtual method used to handle the answers to this type of request, supplying as parameters the values returned by the processing node.

In case no more requests are pending, the `process` method returns `false`. In the implementation of the node manager's `run` method in the `MainNodeManagerImpl` class this is used to determine when the system has finished executing all requests.

Implementation

The `MainNodeManagerImpl` class implements the virtual methods declared by the abstract `MainNodeManager` base class, defining how the manager reacts to answers received from each processing node, as well as the `run` method required by the `NodeManager` class that defines how the master node works in general.

When starting, the master node reads in the main DiCon configuration file for the current run, and then initializes logging to the log files, as given in the configuration, both on the master node and on all processing nodes. After that, the master node

```
1  /// Main node manager interface.
2  class MainNodeManager
3    : public NodeManager
4  {
5  protected:
6    /// Create main node manager.
7    MainNodeManager( boost::mpi::communicator &world );
8
9  public:
10   virtual ~MainNodeManager();
11
12 protected:
13   /// Get minimum child ID.
14   int min_child() const;
15   /// Get maximum child ID.
16   int max_child() const;
17
18   /// Get number of children.
19   unsigned child_count() const;
20
21 protected:
22   void start_logging ( int node, const boost::filesystem::path &logfile, LogLevel min_level         );
23   void init_optimizer( int node
24                      , const boost::filesystem::path &optimizer_logfile
25                      , const boost::filesystem::path &simulator_logfile
26                      , const std::string &optimizer_library, const arguments_t &optimizer_arguments
27                      , const std::string &simulator_command, const arguments_t &simulator_arguments
28                      , const boost::optional<boost::filesystem::path> &optimizer_map_file
29                      , const boost::optional<boost::filesystem::path> &optimizer_lib_file          );
30   void init_simulator( int node
31                      , const boost::filesystem::path &simulator_logfile
32                      , const std::string &simulator_command, const arguments_t &simulator_arguments );
33   void init_combined ( int node
34                      , const boost::filesystem::path &optimizer_logfile
35                      , const boost::filesystem::path &simulator_logfile
36                      , const std::string &optimizer_library, const arguments_t &optimizer_arguments
37                      , const std::string &simulator_command, const arguments_t &simulator_arguments
38                      , const boost::optional<boost::filesystem::path> &optimizer_map_file
39                      , const boost::optional<boost::filesystem::path> &optimizer_lib_file          );
40   void get_policy    ( int node                                                                    );
41   void simulate      ( int node, const policy_t &policy                                            );
42   void update        ( int node, const policy_t &policy, double reward                             );
43   void step_combined ( int node                                                                    );
44   void dump_optimizer( int node, const boost::filesystem::path &file, DumpOptimizerMode mode      );
45   void dump_policies ( int node, const boost::filesystem::path &file, unsigned count, bool display );
46   void shutdown      ( int node                                                                    );
47
48 protected:
49   /// Get node state.
50   NodeState state( int node ) const;
51
52 protected:
53   virtual void finish_start_logging ( int node                                                )=0;
54   virtual void finish_init_optimizer( int node                                                )=0;
55   virtual void finish_init_simulator( int node                                                )=0;
56   virtual void finish_init_combined ( int node                                                )=0;
57   virtual void finish_get_policy    ( int node, const boost::optional<policy_t> &policy       )=0;
58   virtual void finish_simulate      ( int node, double reward                                 )=0;
59   virtual void finish_update        ( int node                                                )=0;
60   virtual void finish_step_combined ( int node, bool got_policy                               )=0;
61   virtual void finish_dump_optimizer( int node                                                )=0;
62   virtual void finish_dump_policies ( int node                                                )=0;
63   virtual void finish_shutdown      ( int node, bool done                                     )=0;
64
65   /// Handle failure while processing request.
66   virtual void handle_failure       ( int node, const std::string &what                       )=0;
67
68   /// Wait for and finish one request.
69   bool process();
70 };
```

Figure 3.2: Interface to the `MainNodeManager` class.

repeatedly asks processing nodes to spawn optimizers and simulators, working its way through the job list defined in the configuration.

Due to the star topology used for communication, each request has to go through the master node. While this may limit scalability in theory, in practice the CPU usage on the master node stayed as low as 10 % or less, even with as many as 512 nodes in use during representative test runs on the TACC computer cluster *Lonestar*.

The processing of jobs on the master node works by stepping through the lazy list defined by the job definitions in the configuration file. The lazy list concept is discussed in more detail in section 3.4.2 on page 37. For each job that is currently active, having at least one processing node working on it, a state machine is kept at the master node. This job state machine is explained in section 3.3 on page 28.

3.2.2 Processing node

The `ProcNodeManager` class defines a set of abstract virtual methods that define how the processing node reacts to the requests it receives from the master node. Each processing node processes one request at a time, and in the order requests are received. This makes implementing the processing node manager as simple as possible, since no multitasking is involved: there is only a single thread of execution.

The interface to the `ProcNodeManager` class is shown in figure 3.3 on the next page. Each abstract virtual function defines a request that the processing node can execute when asked to by the master node.

There are requests to start logging (`start_logging`), to initialize an optimizer (`init_optimizer`), simulator (`init_simulator`), or combined optimizer/simulator object (`init_combined`), to get a policy from an optimizer (`get_policy`), to simulate with a policy (`simulate`), to update the reward value of a policy (`update`), to get a policy, simulate it and update its reward value in a single step at a combined optimizer/simulator node (`step_combined`), to dump the optimizer state (`dump_optimizer`) or best policies (`dump_policies`), and to shut down the processing node (`shutdown`).

Each processing node's `run` method repeatedly waits for an incoming request from the master node; it executes each request with the parameters given, and sends back any return values. Exceptions that are thrown while executing a request are caught and an appropriate answer is sent to indicate the error.

```
1  /// Processing node manager interface.
2  class ProcNodeManager
3    : public NodeManager
4  {
5  protected:
6    /// Create processing node manager.
7    ProcNodeManager( boost::mpi::communicator &world );
8
9  public:
10   virtual ~ProcNodeManager();
11
12 protected:
13   virtual void start_logging ( const boost::filesystem::path &logfile, LogLevel min_level                    )=0;
14   virtual void init_optimizer( const boost::filesystem::path &optimizer_logfile
15                              , const boost::filesystem::path &simulator_logfile
16                              , const std::string &optimizer_library, const arguments_t &optimizer_arguments
17                              , const std::string &simulator_command, const arguments_t &simulator_arguments
18                              , const boost::optional<boost::filesystem::path> &optimizer_map_file
19                              , const boost::optional<boost::filesystem::path> &optimizer_lib_file          )=0;
20   virtual void init_simulator( const boost::filesystem::path &simulator_logfile
21                              , const std::string &simulator_command, const arguments_t &simulator_arguments )=0;
22   virtual void init_combined ( const boost::filesystem::path &optimizer_logfile
23                              , const boost::filesystem::path &simulator_logfile
24                              , const std::string &optimizer_library, const arguments_t &optimizer_arguments
25                              , const std::string &simulator_command, const arguments_t &simulator_arguments
26                              , const boost::optional<boost::filesystem::path> &optimizer_map_file
27                              , const boost::optional<boost::filesystem::path> &optimizer_lib_file          )=0;
28   virtual boost::optional<policy_t>
29               get_policy    (                                                                              )=0;
30   virtual double
31               simulate      ( const policy_t &policy                                                       )=0;
32   virtual void update        ( const policy_t &policy, double reward                                        )=0;
33   virtual bool step_combined (                                                                              )=0;
34   virtual void dump_optimizer( const boost::filesystem::path &file, DumpOptimizerMode mode                 )=0;
35   virtual void dump_policies ( const boost::filesystem::path &file, unsigned count, bool display           )=0;
36   virtual bool shutdown      (                                                                              )=0;
37
38 public:
39   /// Run processing node manager.
40   virtual void run();
41 };
```

Figure 3.3: Interface to the `ProcNodeManager` class.

When the `shutdown` request is received and the implementation of the virtual `shutdown` method returns `true`, the processing node manager is shut down.

Implementation

Since the `run` method is already implemented by the `ProcNodeManager` class, the derived `ProcNodeManagerImpl` class only provides implementations to each of the request-handling methods.

The implementation of these methods is done in a straightforward way: each processing node can hold either an optimizer, a simulator, or a combined optimizer/simulator object of the corresponding class (`Optimizer`, `Simulator`, or `Combined`). When the corresponding `initialize_...` request is received, the current object is destroyed and a new object is created with the parameters given in the request.

The remaining requests are valid only when the object currently loaded is of the appropriate type: `get_policy` and `update` work only on a regular optimizer object, `simulate` works only on a regular simulator object, `step_combined` works only on a combined optimizer/simulator object. Each of the `dump_...` methods works only on an optimizer or combined optimizer/simulator object. If the current object does not match the request, an error is returned.

3.3 Job state machine

For each job that is currently active in the system, in other words for each job that has at least one processing node in the computer cluster working on it, a state machine is kept at the master node.

This state machine indicates how far along the corresponding job is in its processing lifetime. There are four states defined per job: `INIT_OPTIMIZER`, `RUNNING_NORMAL`, `DUMPING_STATES`, and `DUMPING_RESULT`.

Each job starts out in the `INIT_OPTIMIZER` state which is equivalent to initializing a new optimizer or combined optimizer/simulator object on a hitherto unused processing node in the system. As soon as this optimizer or combined optimizer/simulator object has been set up, the job transitions to the `RUNNING_NORMAL` state.

In the `RUNNING_NORMAL` state, the master node repeatedly requests policies from the optimizer, starts simulators on other processing nodes as necessary, and dis-

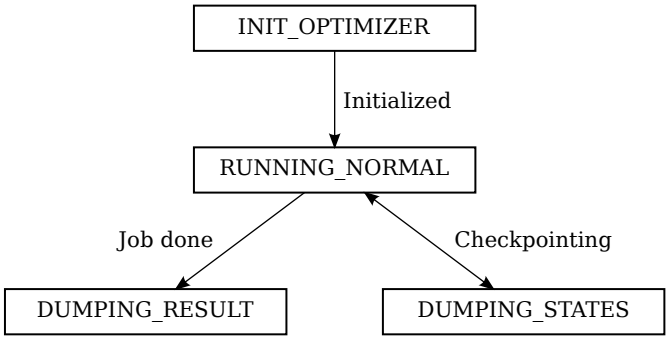


Figure 3.4: State machine used by jobs that are currently active in the system. Each job starts out in the `INIT_OPTIMIZER` state, initializing an optimizer object. When this initialization is done, the job transitions to the `RUNNING_NORMAL` state where it repeatedly simulates policies and updates the optimizer with reward values. When a checkpoint is to be taken, the job transitions to the `DUMPING_STATES` state to dump the optimizer's state; returning back to the `RUNNING_NORMAL` state when this is done. When the end of a job is reached, the job transitions to the `DUMPING_RESULT` state to write out the final results; when this is done, the job is released and a new job is started by DiCon eventually.

tributes policies to these simulators as they become available. Whenever a simulator finishes simulating, the optimizer is updated with the reward value obtained during simulation. If the job is running on a combined optimizer/simulator node, the master node instead repeatedly calls the `step_combined` method.

There are two ways to leave the `RUNNING_NORMAL` state: when the optimizer is not able to provide any more policies, or the maximum number of iterations specified in the configuration has been reached, the job assumes the `DUMPING_RESULT` state. If instead the master node determines that a checkpoint is to be taken, as specified in the configuration, the job assumes the `DUMPING_STATES` state.

In both the `DUMPING_STATES` state and the `DUMPING_RESULT` state, the master node asks the optimizer or combined optimizer/simulator object to dump its internal state to a file, along with the list of best policies obtained by the optimization algorithm. In the `DUMPING_STATES` state the job then transitions back to the `RUNNING_NORMAL` state, whereas in the `DUMPING_RESULT` state the job is released as soon as the last dump action finishes.

A graphical representation of the job state machine is shown in figure 3.4.

3.4 Specifier parser

Often it is useful to run an optimization job with the same simulator program for many different parameters. For example, disease parameters such as the basic reproduction number R_0 or the duration of the infectious period are commonly known only within certain limits. It is therefore useful to investigate how the optimal policy changes for different values.

In order to make it easy to specify many jobs at once in a single DiCon configuration file, a special syntax is offered to declare *sets* of argument values, in contrast to the direct definition of only a single argument value. In the configuration file, this syntax is used for the argument specifiers of both simulator programs and optimization algorithms; a related syntax is used for specifying absolute checkpoint positions.

3.4.1 Specifier grammar

The grammar for the special syntax used to declare sets of arguments and checkpoints is shown in figure 3.5 on the next page, in Extended Backus–Naur Form. The equivalent, albeit more intuitive, representation in the form of syntax or "railroad" diagrams is given in figure 3.6 on page 32.

In the special syntax, braces, or curly brackets, denote lists of alternative character strings, while square brackets denote numerical ranges. When more than one such set specifier is given in a single argument, the set of all possible combinations is generated; this is equivalent to the Cartesian product of the individual sets. The same operation is performed when more than one argument is given for either simulator program or optimization algorithm.

Internally, the recursive descent parser generator framework offered by the Boost Spirit library is used to parse argument specifiers according to the specifier grammar. Boost Spirit makes use of expression templates and template meta-programming, allowing the grammar to be written exclusively in C++, without the need for external parser generators, in contrast to the lexer and parser generator approach traditionally used which requires external tools such as `flex`, `yacc`, or `bison`.

Since the grammar itself defines only the syntax of argument specifiers and checkpoint or schedule specifiers, the following paragraphs describe the semantic meaning of each non-terminal symbol, as defined by the visual grammar representation shown

```
ArgumentSpecifier = Set ;
ScheduleSpecifier = "[" , ( RegularRange | ExpressionRange ) , "]"
                  | Expression , { "," , Expression } ;

Set = Specifier , { "|" , Specifier } ;
Specifier = SpecifierElement , { SpecifierElement } ;
SpecifierElement = "[" , ( RegularRange | ExpressionRange ) , "]"
                 | "{" , Set , "}"
                 | Literal ;

RegularRange = RegularRangeElement , { "|" , RegularRangeElement } ;
RegularRangeElement = Singleton | SimpleRange | ExtendedRange ;

ExpressionRange = Expression , ":" , ( ExpressionSubrange , { ":" , ExpressionSubrange } ) ;
ExpressionSubrange = [ Identifier , "=" ] , RegularRange ;

Singleton = Expression ;
SimpleRange = Expression , ".." , [ Expression ] ;
ExtendedRange = Expression , "," , Expression , ".." , [ Expression ] ;

Expression = Term , { ( "+" | "-" ) , Term } ;
Term = Factor , { ( "*" | "/" ) , Factor } ;
Factor = Group , "^" , Factor
       | [ "+" | "-" ] , Group ;
Group = "(" , Expression , ")"
      | FunctionCall | Identifier | Real ;
FunctionCall = FunctionName , "(" , Expression , ")" ;
Identifier = ( Letter | "_" ) , { Letter | Digit | "_" } ;

Literal = LiteralElement , { LiteralElement } ;
LiteralElement = "'" , { SingleQuoted } , "'"
               | '"' , { DoubleQuoted } , '"'
               | RegularCharacter ;
SingleQuoted = ? any character except ' ? ;
DoubleQuoted = ? any character except " ? ;
RegularCharacter = ? any character except ', ", |, [, ], {, } ? ;

Real = [ "+" | "-" ] , Digits , [ "." , Digits ] , [ ( "e" | "E" ) , [ "+" | "-" ] , Digits ] ;
Digits = Digit , { Digit } ;

Digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
Letter = ? any character either in range "a" to "z", or "A" to "Z" ? ;
FunctionName = "sqrt"  | "round"  | "trunc" | "floor" | "ceil"  | "abs"
             | "log"   | "log2"   | "log10" | "exp"   | "exp2"  | "exp10"
             | "cos"   | "sin"    | "tan"   | "acos"  | "asin"  | "atan"
             | "cosh"  | "sinh"   | "tanh"  | "acosh" | "asinh" | "atanh"
             | "erf"   | "erfc"   | "j0"    | "j1"    | "y0"    | "y1"
             | "gamma" | "lgamma" | "log1p" | "expm1" ;
```

Figure 3.5: Grammar for the argument and schedule specifiers used in the configuration file for each DiCon run, with start symbols `ArgumentSpecifier` and `ScheduleSpecifier`, respectively. The grammar is given in Extended Backus–Naur Form, as defined in the international standard ISO/IEC 14977:1996 (E) [30].

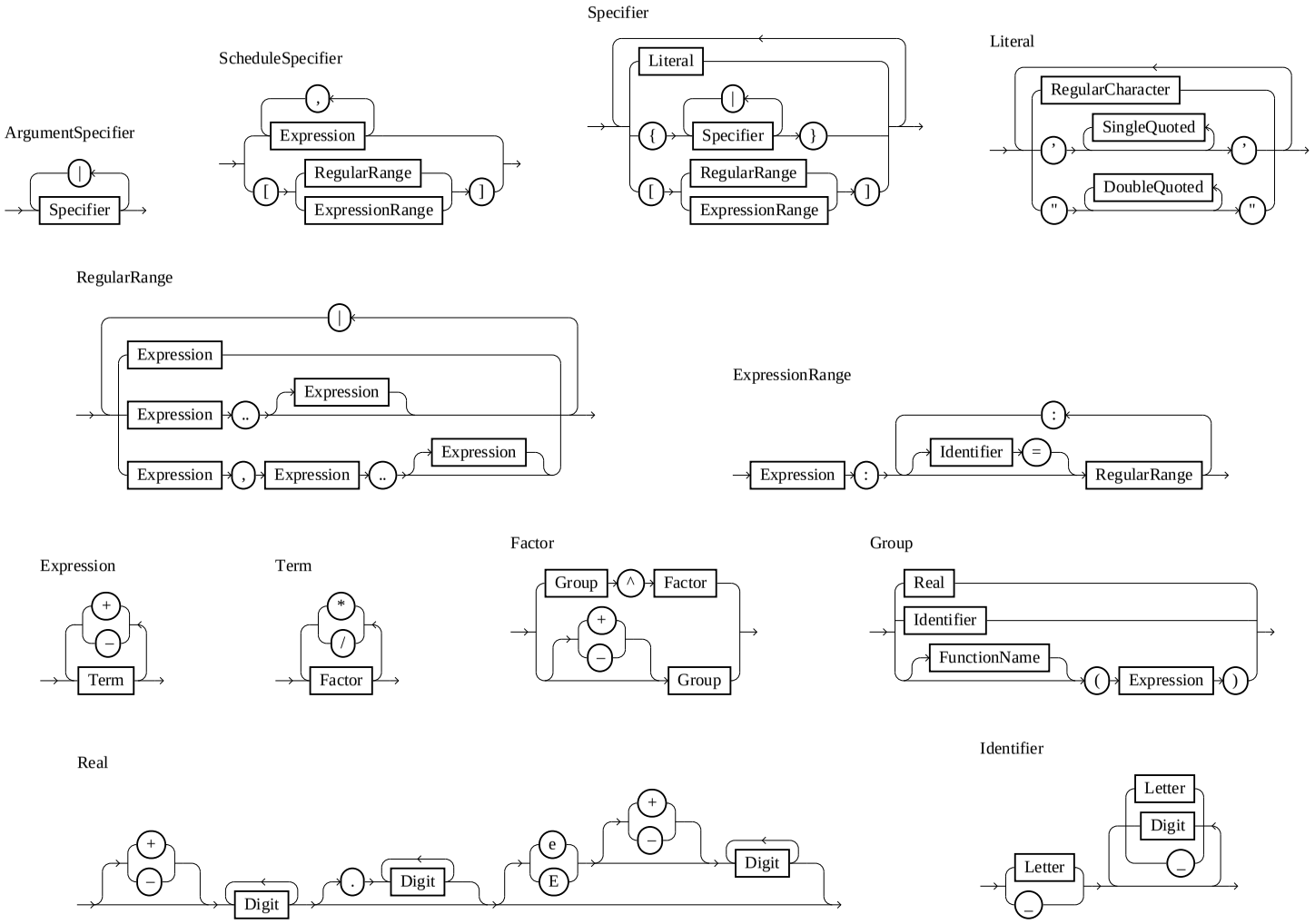


Figure 3.6: Visual rendering of most parts of the grammar for argument and schedule specifiers, with start symbols `ArgumentSpecifier` and `ScheduleSpecifier`, respectively, in the form of syntax or "railroad" diagrams. Compare to the textual rendering of this grammar in Extended Backus–Naur Form given in figure 3.5 on the preceding page for the definitions of the missing symbols `Letter`, `Digit`, `RegularCharacter`, `SingleQuoted`, `DoubleQuoted`, and `FunctionName`.

in figure 3.6 on the previous page.

The type of each symbol is given after the symbol name, with the notation $[\cdot]$ denoting lists of the given type, and $\langle\cdot\rangle$ denoting a lazy expression of the given type; lazy expressions are explained in detail in section 3.4.2 on page 37. **string** is an arbitrary string of characters, **double** is a real number.

`Real` : **double** Real number with the given representation in scientific notation. In this notation, a string of the form $\pm$`A.BeC` is equivalent to the real number with value $\pm$`A.B` $\times 10^{\texttt{C}}$. Leading and trailing zeros are not treated in any special way.

Example `-1.23e3` returns the real number $-1.23 \times 10^3 = -1230$.

`Identifier` : **string** String equivalent to the given string. This is used to denote variable names. In order to make those names distinguishable from real numbers, identifiers must not begin with a digit.

Example `x` returns the string `x`, and `_a2` returns the string `_a2`.

`Group` : $\langle$**double**$\rangle$ Lazy expression that has either (a) the value of the given real number, or (b) the value of the variable with the given identifier, or (c) the value of the application of the function with the given name to the value of the given lazy expression, or (d) the value of the given lazy expression itself. Functions are defined by their counterpart with the same name in the Standard C Math library[2], with the exceptions that "abs" refers to `fabs`, and "gamma" refers to `tgamma`.

Example `1.23` returns the lazy expression 1.23, `x` returns the lazy expression x, `sin(1+x)` returns the lazy expression $\sin(1 + x)$, and `(x^2+y^2)` returns the lazy expression $x^2 + y^2$.

`Factor` : $\langle$**double**$\rangle$ Lazy expression that has either (a) the value of the exponentiation obtained from the values of the given two lazy expressions, or (b) the negative

[2]The Standard C Math library is included with `#include <math.h>` or `#include <cmath>` in C and C++, respectively.

value of the given lazy expression, or (c) the value of the given lazy expression itself.

Example `(x+y)^2` returns the lazy expression $(x+y)^2$, `x^y^z` returns the lazy expression $x^{(y^z)}$, `-x` returns the lazy expression $-x$, and both `+x` and `x` return the lazy expression x.

`Term` : ⟨**double**⟩ Lazy expression that has the value obtained from the series of left-associative multiplications and divisions of the values of the given lazy expressions.

Example `x` returns the lazy expression x, `x*2` returns the lazy expression $2x$, `x/y/z` returns the lazy expression $(x/y)/z$, or equivalently $x/(yz)$, and `1/3*4` returns the lazy expression $4/3$.

`Expression` : ⟨**double**⟩ Lazy expression that has the value obtained from the series of left-associative additions and subtractions of the values of the given lazy expressions.

Example `x` returns the lazy expression x, `x+2` returns the lazy expression $x+2$, `1-3+4` returns the lazy expression 2, and `2-3*x` returns the lazy expression $2-3x$.

`RegularRange` : ⟨[**double**]⟩ Lazy list that contains the following elements. Let X, Y, and Z denote the values of the first, second, and third given lazy expression, respectively. A part of the lazy list is then either (a) the single element X, or (b) the infinite list of values with step size 1 starting with X, or (c) the list of values with step size 1 starting with X and being less than or equal to Y, or (d) the infinite list of values with step size $Y-X$ starting with X, or (e) the list of values with step size $Y-X$ starting with X and being less than or equal to Z (for positive step size) or being greater than or equal to Z (for negative step size).

If more than one list specifier, separated by the terminal `|`, is given, the lazy set union of the corresponding ranges is returned. For this, each individual range is sorted; the lazy list returned will then also be sorted. It is an error to specify an infinite range with negative step size when more than one list specifier is given.[3]

[3]Such a list cannot be sorted since it has no smallest element; but this is necessary to obtain the

As indicated by its type, `RegularRange` contains real numbers, *not* lazy expressions that would only evaluate to real numbers: `RegularRange` itself is lazy, its elements are not. When parsing this non-terminal, all lazy expressions used therein are evaluated in the context given by the `ExpressionRange` containing this `RegularRange` non-terminal. If `RegularRange` is used outside `ExpressionRange` it must not contain any `Identifier` non-terminals.

Example `4` returns the lazy list $[4]$, `1..10` returns the lazy list $[1, 2, \ldots, 10]$, `1/4,1/2..1` returns the lazy list $[0.25, 0.5, 0.75, 1]$, `5,4..` returns the infinite lazy list $[5, 4, \ldots]$, `4..7|9,8..2` returns the lazy list $[2, 3, 4, 5, 6, 7, 8, 9]$, and `0,3..|0,7..` returns the infinite lazy list $[0, 3, 6, 7, 9, 12, 14, 15, \ldots]$.

`ExpressionRange` : [**double**] List that contains the following elements. Each of the `RegularRange` non-terminals can be assigned a variable name, given by a corresponding `Identifier` non-terminal; for each range that is not accompanied by a name, one will be picked from the list $\{\mathtt{x}, \mathtt{y}, \mathtt{z}, \mathtt{a}, \ldots, \mathtt{w}\}$, in that order. It is an error when the same name is used by more than one range, or when more than 26 ranges without name are specified.

From left to right, the lazy list given by each `RegularRange` non-terminal is evaluated. For each element in the current list, the lazy lists to the right are evaluated recursively; these lazy lists may contain variable references to the variable associated with the current lazy list; the value of this variable will be the element that is currently being processed. When the final range has been reached in this recursive fashion, the lazy value given by the `Expression` non-terminal is evaluated, with all variables associated with one of the given ranges set to their current values. The value returned from this evaluation is then added to the list resulting from the `ExpressionRange` non-terminal.

If at least one of the ranges used within `ExpressionRange` evaluates to an infinite list, `ExpressionRange` will also evaluate to an infinite list. It is an error when the lazy value given by `Expression` contains a reference that is not defined by one of the ranges given by `RegularRange`; it is also an error when one of the `RegularRange` non-

---

sorted set union.

terminals contains a reference that is not defined by a `RegularRange` non-terminal to its left. It is an error when a variable is defined that is used neither by a range to its right nor by the `Expression` non-terminal; this is enforced to make sure that a reference is not accidentally forgotten.

Example `2*x:1..3` returns the list $[2, 4, 6]$, `x+z:1|9:z=10,20..30` returns the list $[11, 21, 31, 19, 29, 39]$, and `z*10-1:x=1..5:y=1..x:z=10*x+y` returns the list $[109, 209, 219, 309, 319, 329, 409, 419, 429, 439, 509, 519, 529, 539, 549]$.

`ScheduleSpecifier` : [**double**] List that contains either (a) the list of values given by the `Expression` non-terminals, sorted in increasing order, with all duplicates removed, and each value rounded to the nearest integer, or (b) the values of the list given by the `ExpressionRange` non-terminal, with each value rounded to the nearest integer, or (c) the values obtained by evaluating the lazy list given by the `RegularRange` non-terminal, in the context of an empty set of variables, with each value rounded to the nearest integer.

It is an error when one of the given expressions contains references to a variable. It is also an error when the values returned by either range are not in strictly increasing order, or when one of the rounded values of the elements of the list resulting from the `ScheduleSpecifier` is less than 1 or greater than $2^{64} - 1$ ($\approx 1.84 \times 10^{19}$).

Example `5,1,9,1,5,3` returns the list $[1, 3, 5, 9]$, and `[0.9+2^x:0..]` returns the infinite list $[2, 3, 5, 9, 17, \ldots]$.

`Literal` : [**string**] List that contains the single entry that is given by the given string of characters. Characters given by `RegularCharacter` are used directly; single and double quotation marks used to denote `SingleQuoted` and `DoubleQuoted`, respectively, are removed.

Example `a"|"b'"'c` returns the list [`a|b"c`].

`Specifier` : [**string**] List that contains the combination, or Cartesian product, of the elements of all inner lists. The terminals `[...]` denote a list, or a lazy list evaluated in the context of an empty set of variables, of **double** elements which are

converted to their string representations in scientific notation. The terminals {...} denote the concatenation of all the inner `Specifier` lists. The list given by the `Literal` non-terminal is used directly.

It is an error when a `RegularRange` non-terminal within `Specifier` contains references to any variables.

Example `a{b|c}` returns the list [`ab`, `ac`], `[1..2][8..9]{a|b}` returns the list [`18a`, `18b`, `19a`, `19b`, `28a`, `28b`, `29a`, `29b`], and `{[1..3]x|{a|b}{c|d}}` returns the list [`1x`, `2x`, `3x`, `ac`, `ad`, `bc`, `bd`].

`ArgumentSpecifier` : [**string**] List that contains the concatenation of all the inner `Specifier` lists.

Example `a|b|[1..3]` returns the list [`a`, `b`, `1`, `2`, `3`].

3.4.2 Lazy lists and values

Lazy lists are special kinds of lists whose list elements have not been preallocated in memory; instead, elements are only created "on the fly" as they are being requested. For this reason, lazy lists use up significantly less memory compared to ordinary lists, especially when those lists contain many elements.

Similarly, lazy values represent expressions that indicate how a given value can be calculated, in contrast to storing the value itself. This is necessary whenever the value in question depends on some other, external parameter.

Lazy values are used explicitly in the definition of the semantic meaning of argument and schedule specifiers, as given in section 3.4.1 on page 30, to describe the elements of "expression ranges" where an arithmetic expression is applied to each value within a given range.

DiCon implements a very simple interface for lazy values, defining the following two methods: `references` returns a list of variable identifiers that are used within the lazy expression, and `evaluate` returns the value for a given variable assignment; see figure 3.7 on the following page for the actual definitions. Through the use of virtual methods and inheritance a number of classes are derived that implement the lazy application of functions to one or more lazy values.

```
1  /// Lazy value.
2  template< typename T >
3  class LazyValue {
4  public:
5    /// Value type.
6    typedef T value_t;
7    /// Pointer to lazy value.
8    typedef std::auto_ptr<LazyValue> ptr_t;
9    /// Set of variable names.
10   typedef std::set<std::string> references_t;
11   /// Variable assignment.
12   typedef std::map<std::string, T> variables_t;
13
14 public:
15   virtual ~LazyValue() {}
16
17 public:
18   /// Get list of references.
19   virtual references_t references() const = 0;
20
21 public:
22   /// Evaluate lazy value.
23   virtual T operator()( const variables_t &variables ) const = 0;
24 };
```

Figure 3.7: Interface to the `LazyValue` class.

```
1  /// Lazy set.
2  template< typename T >
3  class LazySet {
4  public:
5    /// Set value type.
6    typedef T value_t;
7    /// Pointer to lazy set.
8    typedef std::auto_ptr<LazySet> ptr_t;
9
10 public:
11   virtual ~LazySet() {}
12
13 public:
14   /// Clone lazy set.
15   virtual ptr_t clone() const = 0;
16
17 public:
18   /// Reset lazy set.
19   virtual void reset() = 0;
20
21 public:
22   /// Check if set has element.
23   virtual bool has() const = 0;
24   /// Get current element in set.
25   virtual T get() const = 0;
26   /// Skip current element in set.
27   virtual void inc() = 0;
28 };
```

Figure 3.8: Interface to the `LazySet` class.

The lazy lists interface has four methods: `reset` returns the list's internal iterator to the beginning of the list, `has` checks if the list contains another element, `get` returns the element at the list's current position, and `inc` advances the iterator to the next element in the list; see figure 3.8 on the previous page for the actual definitions. The additional method `clone` creates an exact, independent copy of the original lazy list. Lazy lists do not have a predefined length and can therefore be infinite; this is used for schedule specifiers that describe when checkpoints are to be taken.

Similar to the implementation of lazy values, lazy lists use virtual methods and inheritance to define a number of derived classes that lazily transform a list, or lazily combine two or more lists.

Lazy lists are used in place of regular lists for every non-terminal described in the semantics of the specifier grammar in section 3.4.1 on page 30; in fact, lazy lists are used for the description of every single job definition within a DiCon run. This is done to avoid the highly redundant alternative of creating each element within an argument specifier list in advance: when defining several thousand arguments of moderate length, the preallocation of all elements might easily take up many megabytes, unnecessarily wasting resources and limiting the number of job definitions per DiCon run to a few million.[4] The approach of using a lazy job queue implementation also allows for the creation of unbounded lists used in checkpoint schedules where the alternative of picking an arbitrary upper bound might not always be feasible.

Lazy lists

The following classes implement the lazy list interface; see figure 3.8 on the previous page for details. These specializations provide functionality that is used to implement the operations on lists required by the argument and schedule specifier grammar as described in section 3.4.1 on page 30.

In the description of each class, $\langle\cdot\rangle$ denotes a lazy list, while $[\cdot]$ denotes a regular list or list-like container. Functors are indicated by $\cdot \to \cdot$. T, S, S_1, and S_2 are arbitrary type variables, used to denote the types of lazy lists and functors.

[4] This may sound a lot but doing optimization runs for only a couple of different parameters that may be tuned independently can easily lead to millions of different combinations. Not being able to run this many parameters because of an out-of-memory error due to wasteful resource management when preparing the job queue would be quite counterintuitive.

`LazyCombine` : $\langle S_1 \rangle \times \langle S_2 \rangle \times (S_1 \times S_2 \to T) \to \langle T \rangle$ Lazy list that is the combination, or the Cartesian product, of the two lazy lists given on construction. The given functor is applied to each pair of elements from the two original lists, producing the elements of the resulting combination list.

`LazyCompose` : $\langle S \rangle \times (S \to \langle T \rangle) \to \langle T \rangle$ Lazy list that is the composition of the given functor with the elements of the given lazy list. The functor is applied in sequence to the elements of the original list, returning a lazy list for each element. The resulting lazy list is the concatenation of the lazy lists returned by the functor.

`LazyConcat` : $\langle T \rangle \times \langle T \rangle \to \langle T \rangle$ Lazy list that is the concatenation of the two lazy lists given on construction.

`LazyContainer` : $[T] \to \langle T \rangle$ Lazy container list that contains exactly the elements of the given container, in the same order as the container returns them.

`LazyEmpty` : $\emptyset \to \langle T \rangle$ Lazy empty list that does not contain any elements.

`LazyRange` : $T \times T \times T \to \langle T \rangle$ Lazy range list that represents the finite (bounded) range with elements $a, a + x, a + x + x, \ldots, b$, or the infinite (unbounded) range with elements $a, a + x, a + x + x, \ldots$, depending on whether b was given on construction or not. a, b, and x are the lower bound, upper bound, and step size, respectively, used when creating the lazy list.

`LazySingleton` : $T \to \langle T \rangle$ Lazy singleton list that contains only the single element given on construction.

`LazyTransform` : $\langle S \rangle \times (S \to T) \to \langle T \rangle$ Lazy list that is the transformation of the given lazy list by the given functor. The functor is applied in sequence to the elements of the original list, producing the elements of the resulting transformed list.

`LazyUnion` : $\langle T \rangle \times \langle T \rangle \to \langle T \rangle$ Lazy list that is the set union of the two lazy lists given on construction. Both original lists must be sorted and may not contain an

element more than once each; the resulting list will then also be sorted and free of duplicates.

Lazy values

The following classes implement the lazy value interface; see figure 3.7 on page 38 for details. These specializations provide functionality that is used to implement the operations on expressions required by the argument and schedule specifier grammar as described in section 3.4.1 on page 30.

As in the description of lazy lists given above, functors are indicated by $\cdot \to \cdot$, while T is an arbitrary type variable, used to denote the type of the lazy value. $\langle\cdot\rangle$ signifies a lazy value, and **string** is a string of characters, used for variable identifiers.

`LazyBinary` : $\langle T\rangle \times \langle T\rangle \times (T \times T \to T) \to \langle T\rangle$ Lazy value that is the lazy application of the given binary functor to the two lazy values given on construction. The resulting lazy value depends on all variables the original lazy values depend on.

`LazyConstant` : $T \to \langle T\rangle$ Lazy value that is constant and always evaluates to the value given on construction. The resulting lazy value does not depend on any variable.

`LazyUnary` : $\langle T\rangle \times (T \to T) \to \langle T\rangle$ Lazy value that is the lazy application of the given unary functor to the lazy value given on construction. The resulting lazy value depends on all variables the original lazy value depends on.

`LazyVariable` : **string** $\to \langle T\rangle$ Lazy value that evaluates to the current assignment of the variable with the name given on construction. The resulting lazy value depends only on this variable.

Chapter 4

Application

This chapter shows how DiCon can be employed to address real-world problems.

4.1 Distribution of antivirals

During the 2009 flu pandemic, caused by swine-origin influenza A virus subtype H1N1, the primary modes of control prior to the large-scale distribution of an effective vaccine included: careful surveillance, social distancing and hygiene measurements, strategic school closures, other community measures, and the prudent use of antiviral medications, used both for prophylaxis, preventing infection, and for treatment, reducing the severity and duration of symptoms. [31]

An earlier version of the DiCon framework was used to investigate the following question: What would be the optimal way of distributing the U. S. strategic national stockpile of antivirals for treatment of infected cases during the early stages of a pandemic, prior to the wide availability of vaccines—in other words: when, and where, should antivirals be released in order to minimize the total number of infections during the spread of the disease?

4.1.1 Model

In the approach to the question of how to distribute antivirals the following epidemic model was devised. The 100 largest metropolitan areas in the United States were selected to form a network, as shown in figure 4.1 on the next page, with edges

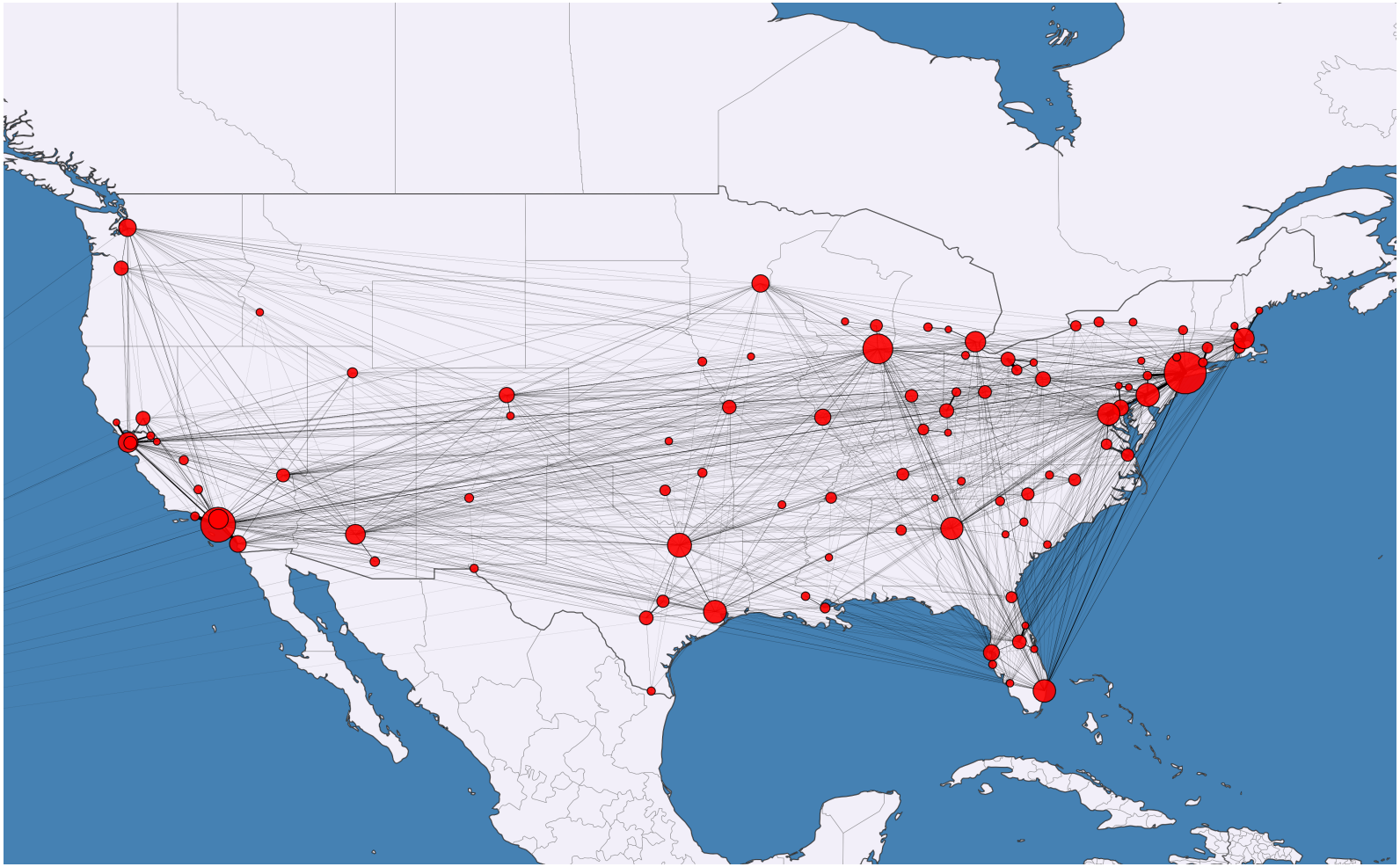


Figure 4.1: Map of the U. S. network used for disease simulation. Red circles represent the 100 largest metropolitan areas, where the area of each circle is proportional to the number of individuals living in the corresponding metropolitan area. Edges represent the movement between metropolitan areas by airline travel and commuter traffic, with line thickness being proportional to the average number of travelers going in either direction per day. [31]

Parameter	Symbol	Value	References
Reproduction number	R_0	1.6, $(1.1, 2.1)$	[32, 33]
Average latency period	L	3 days	[34]
Average asymptomatic infectious period	I_A	2 days	[32]
Average total infectious period (asymptomatic + symptomatic)	I	6 days	[34]
Average infectious period prior to effective antivirals treatment	T	2 days	
Antiviral efficacy	ε	80 %	[35–41]
Local stockpile half-life (wastage)	W	2 months	
Antiviral uptake	U	$(0, 1)$	

Table 4.1: Disease parameters used in the simulation of the 2009 flu pandemic.

between cities representing movement due to airline travel and commuter traffic. Within each of these areas, disease transmission was simulated using a compartmental model with five compartments, using one compartment each for susceptible, exposed, asymptomatic infectious, symptomatic infectious, and recovered individuals. [31]

The actual policy space was very similar to the one shown in figure 1.1(a) on page 4: at the beginning of each month, a certain amount of the remaining available stockpile of antivirals could be distributed among all cities in the network, in proportion either to the number of individuals living in each city or to the current disease prevalence within each city. This distribution step was repeated for 12 months, or until the available stockpile was depleted. [31]

The distributed antivirals would be used immediately for treating symptomatic infectious individuals, significantly reducing the duration of their infectious period. The disease parameters that were used in the simulation are shown in table 4.1.

4.1.2 Results

When not distributing any antivirals, the expected cumulative number of cases after 12 months was nearly 128 million. [31] Several optimized time-based intervention policies were found that were able to decrease this number significantly, depending on the antiviral uptake—this parameter controlled the fraction of the symptomatic infectious individuals that actually sought medical treatment.

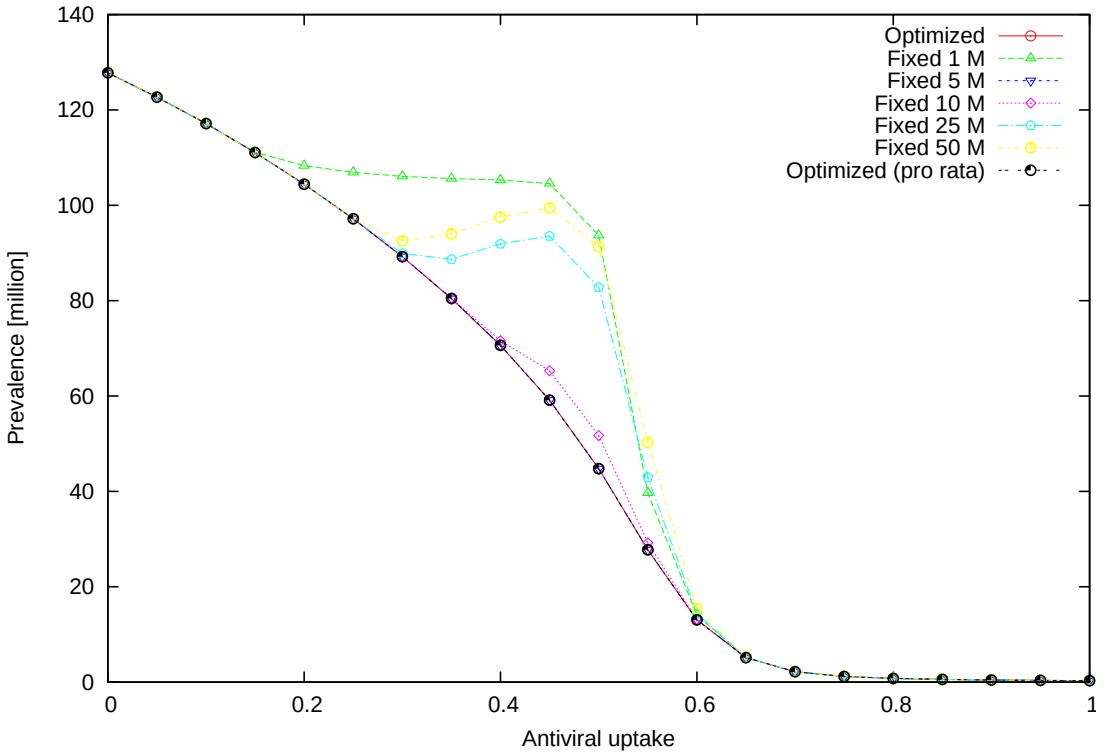


Figure 4.2: Performance of the optimized time-based intervention policies, showing the total expected prevalence, which is the number of individuals getting sick during the first 12 months in the simulation, for different antiviral uptake values; the uptake parameter controlled the fraction of the symptomatic infectious individuals that actually sought medical treatment. The optimized policy is compared to five fixed strategies that distributed either 1, 5, 10, 25, or 50 million doses of antivirals each month. The optimized policy was allowed to distribute antivirals each month either pro rata or in proportion to the current prevalence within each city in the network, the "pro rata" policy was allowed to distribute antivirals pro rata only.

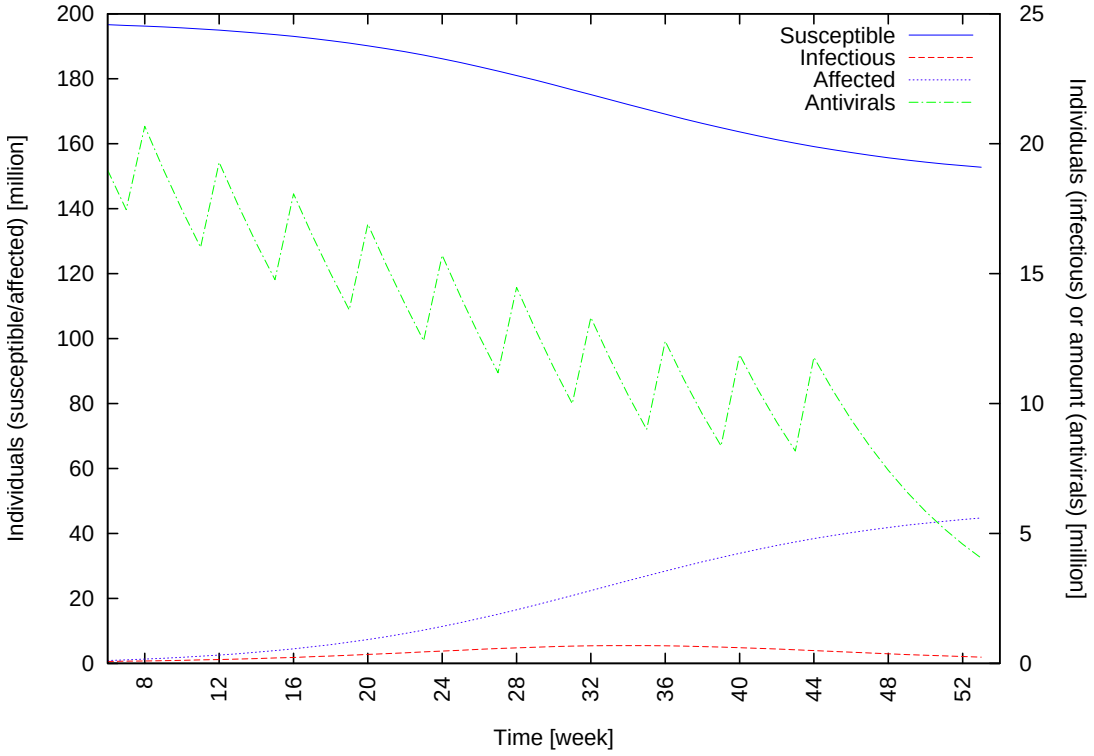


Figure 4.3: Representative plot showing the epidemic curves for susceptible and infectious individuals for a single simulation run. The number of affected individuals is the cumulative number of cases, or the final total prevalence. The antivirals plot curve shows the total amount of antivirals available to be used for medical treatment; the steps are the distribution happening at fixed amounts of 5 million doses each month after the initial release of 31 million doses at the beginning of the simulation; the decrease in-between steps is due to the uptake of antivirals by the (initially exponentially) growing number of infected individuals. The plot was generated for an uptake value of 0.5 where half of all symptomatic infectious individuals sought medical treatment.

Figure 4.2 on page 45 shows the performance of the time-based intervention policies that were found when running the optimization. Two optimized policies are shown: the first was allowed to distribute antivirals each month either pro rata or in proportion to the current prevalence within each city in the network, the second was allowed to distribute antivirals pro rata only.

In addition to the optimized policies, the figure also shows five simple strategies that distributed a fixed amount of antivirals each month, either 1, 5, 10, 25, or 50 million doses, until the available stockpile was depleted.

Surprisingly, the simple "5 M" policy performed virtually as well as both optimized policies, at least for a simulation horizon of 12 months, a local stockpile half-life of 2 months, and a reproduction number of 1.6. The epidemic curves for this simple strategy are shown in figure 4.3 on the previous page.

The observation that a very simple policy, distributing a fixed amount of antivirals each month, was able to perform virtually as well as the optimized policies implies that complicated release strategies obtained from optimization, as shown in figure 4.4 on the following page, or figure 4.5 on page 49, are not necessarily required to reach optimal or near-optimal results. This is good news for public-health decision makers: instead of having to follow a highly complex distribution schedule, a very simple and predictable policy could be used, facilitating the logistics of this distribution.

Another observation is that both optimized policy types performed equally well: there was virtually no difference between an optimized policy that was only allowed to distribute antivirals pro rata, and one that was also allowed to distribute antivirals in proportion to the current prevalence within each city in the network. While the latter strategy releases antivirals more focused on where they might actually be needed, this approach is not required to reach optimal results, at least not when the disease has already spread past its initial introduction.[1] This result also makes it easier for public-health decision makers to coordinate the release of antivirals, since a distribution in proportion to local prevalence would likely be both politically and logistically difficult.

[1]When the disease is well past the initial stage of introduction, and has already spread to all major cities in the network, the two types of distribution become more equivalent anyway: the local relative prevalences within the cities in the network become more similar, so that a distribution in proportion to local prevalence is, in effect, the same as a distribution in proportion to city size.

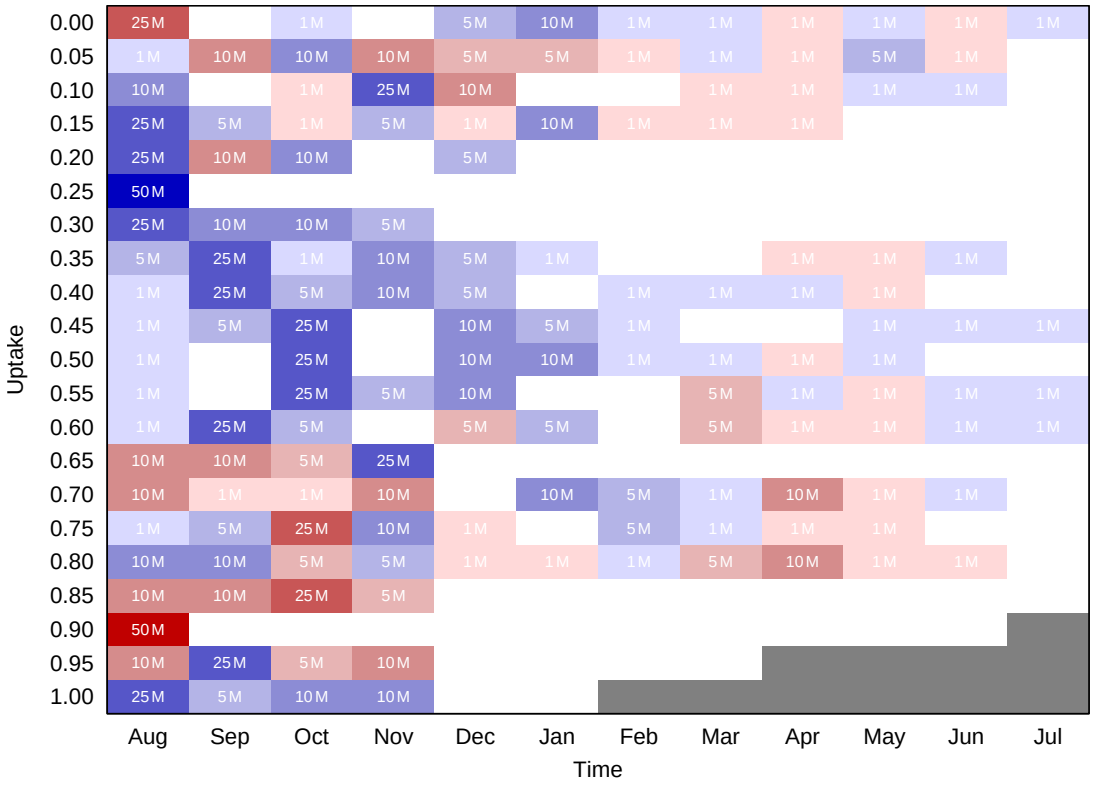


Figure 4.4: Optimized policies derived at different uptake values.[31] The number in each field shows the amount of antivirals scheduled to be distributed during that month, in millions, with August being equivalent to week 8 into the simulation; the color of each field indicates whether the distribution was to be done pro rata (blue), or in proportion to the current prevalence within each city in the network (red). Gray fields indicate that no infected individuals were present during that month in any simulation run—the epidemic had always died out by that point.

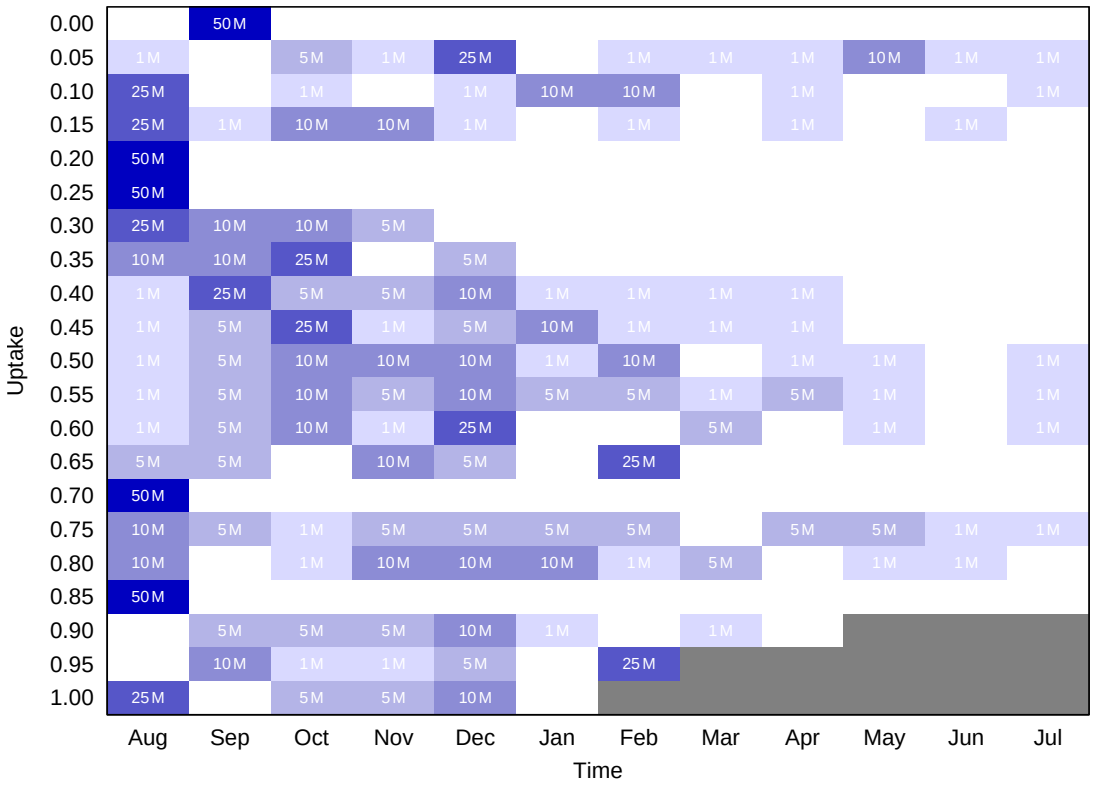


Figure 4.5: Optimized policies derived at different uptake values, where distribution was allowed to happen pro rata only. [31] The number in each field shows the amount of antivirals scheduled to be distributed that month, in millions, with August being equivalent to week 8 into the simulation; the blue color of each field indicates that the distribution was always to be done pro rata. Gray fields indicate that no infected individuals were present during that month in any simulation run—the epidemic had always died out by that point.

Chapter 5

Conclusion

In the preceding chapters, the Disease Control System DiCon was introduced. The central design goals—ease of use, flexibility, performance, and simplicity—were stated, together with descriptions of how these design goals were met in the actual implementation of the system as shown in this work.

An introduction to distributed computing, including a brief look at the history of this branch of computer science, was given, and the use of the distributed computing concept in DiCon was illustrated. A discussion of the optimizer and simulator interfaces was used to highlight the simplicity of these communication means and to show their flexibility in allowing users of DiCon to implement custom optimization algorithms and simulator programs.

The implementations of the master and processing node managers, running the actual DiCon program on the individual nodes in the computer cluster, were outlined, with a particular focus on the mechanics of the job state machine that manages the invocation of optimization jobs in the master node and the assignment of optimizer and simulator roles among the processing nodes in the cluster.

The special syntax that makes defining huge sets of optimization jobs easy was introduced. The formal grammars of these so-called argument and schedule specifiers were given in detail and the facilities that parse these grammars were sketched.

Classes implementing the concepts of lazy lists and values were investigated, making it possible to parse argument and schedule specifiers lazily and allowing the creation of the resulting sets to happen in a memory efficient way.

Finally, a real-world example was given, using an early version of DiCon, illus-

trating the general approach implemented by the Disease Control System: to search for an optimal policy in a given policy space in order to maximize some goal, such as minimizing the number of infections caused by a widespread epidemic. Hints at how this might improve future public-health decision making were given.

Appendix A

Configuration

In the following sections, the options available for use in DiCon's main configuration file are described. An INI file format is used: the configuration file consists of multiple sections, with name/value pairs following each section header. Sections are denoted by their name in square brackets (`[...]`), while names are separated from values by an equal sign (`=`). Lines beginning with a semicolon (`;`) or pound sign (`#`) are ignored and can be used for comments.

An exemplary configuration file is shown in figure A.1 on the following page.

A.1 `[global]` section

The `[global]` section contains options that influence the general behavior of the system, not necessarily related to any specific optimization job. The following options are defined, with default values given in parentheses. The expression "working directory" is used when referring to the directory that is given as command-line argument when starting DiCon.

`main_logfile` **(default:** `log/main.log`**)** Name of the main log file used for messages originating from the master node. The file name is interpreted relative to the given working directory. Missing directories are created automatically. If a file with the given name already exists, alternative file names of the form *main_logfile*`.1`, *main_logfile*`.2`, *main_logfile*`.3`, etc. are tried until a free file name is found.

```
; Sample DiCon configuration file.
;
; Lines starting with ';' are comments.
; The options below show default values.

[global]
;main_logfile=log/main.log
;main_log_level=info
;node_logfile=log/node_%04u.log
;node_log_level=info
;max_jobs=1000000
;job_dir=job-%u
;arg_dir=argument-%05u
;job_logfile=job.log
;job_log_level=info
;checkpoint=checkpoint.xml
;opt_logfile=log/optimizer-node_%04u.log
;sim_logfile=log/simulator-node_%04u.log
;optimizer_map_file=dump/%012u-optmap.bin
;optimizer_lib_file=dump/%012u-optlib.bin
;policy_bin_dumpfile=dump/%012u-policy.bin
;policy_txt_dumpfile=dump/%012u-policy.txt
;policy_count=10
;policy_bin_result=result-policy.bin
;policy_txt_result=result-policy.txt

main_log_level=debug
node_log_level=debug
job_log_level=debug

[job-1]
;max_sims=0
;max_nodes=0
;job_backlog=1
;node_backlog=1
;checkpoint_secs=14400
;checkpoint_sims=0
;checkpoints=
;optimizer=
;simulator=
;opt_args[1]=
;sim_args[1]=

max_nodes=0
job_backlog=1
node_backlog=1
checkpoint_secs=0
checkpoint_sims=0
optimizer=../optimizer/exhaustive.so
simulator=../simulator/simple_simulator.py

[job-2]
; Some other job set here.
```

Figure A.1: Configuration file. An INI file format is used: configuration files have sections, with name/value pairs following section headers. Sections are denoted by their name in square brackets (`[...]`), while names are separated from values by an equal sign (`=`). Lines beginning with a semicolon (`;`) or pound sign (`#`) are ignored and can be used for comments.

`main_log_level` **(default:** `info`**)** Log level used for the main log file. Valid values are `debug`, `info`, `warning`, `error`, `failure`. Lower log levels include higher ones, so using a setting of `info` will include warnings, errors, and failures as well.

`node_logfile` **(default:** `log/node_%04u.log`**)** Name mask for the log files used for messages originating from processing nodes. This mask contains a single `printf`-placeholder that is replaced by the number of the processing node in the cluster. The file names are interpreted relative to the given working directory. Missing directories are created automatically. If a file with the given name already exists, alternative file names are tried until a free name is found; see `main_logfile` for details.

`node_log_level` **(default:** `info`**)** Log level used for the node log files. Valid values are `debug`, `info`, `warning`, `error`, `failure`; see `main_log_level` for details.

`max_jobs` **(default:** `1000000`**)** Maximum number of jobs allowed in the configuration file. This value is used for detecting infinite job sets. It is an error to define more than this many jobs in the `[job-...]` sections.

`job_dir` **(default:** `job-%u`**)** Name mask for the directory used for each job. This mask contains a single `printf`-placeholder that is replaced by the number of the corresponding `[job-...]` section. The directory name is interpreted relative to the given working directory. Missing directories are created automatically.

`arg_dir` **(default:** `argument-%05u`**)** Name mask for the directory used for argument sets defined within each job section. This mask contains a single `printf`-placeholder that is replaced by the number of the argument set within the enclosing `[job-...]` section. The directory name is interpreted relative to the corresponding `job_dir` directory. Missing directories are created automatically. All additional files relating to this job and argument set are put into the directory *job_dir*`/`*arg_dir*.

Job-related

All of the following options define file names or settings related to the processing of a single job and argument set. They are defined in the `[global]` section, and not in

the `[job-...]` sections, in order to make it easier to run batch jobs on the output of an entire DiCon run: enforcing the same naming scheme inside each *job_dir*/*arg_dir* directory makes it easier to run repetitive tasks. All of the file names are interpreted relative to the *job_dir*/*arg_dir* directory of the corresponding job and argument set.

`job_logfile` **(default:** `job.log`**)** Name of the log file used for messages relating to an optimization job and argument set. The file name is interpreted relative to the corresponding *job_dir*/*arg_dir* directory. Missing directories are created automatically. If a file with the given name already exists, alternative file names are tried until a free name is found; see `main_logfile` for details.

`job_log_level` **(default:** `info`**)** Log level used for the job log files. Valid values are `debug`, `info`, `warning`, `error`, `failure`; see `main_log_level` for details.

`checkpoint` **(default:** `checkpoint.xml`**)** Name of the file used for the current state of each job. The checkpoint file is in Extensible Markup Language (XML) format. The file name is interpreted relative to the corresponding *job_dir*/*arg_dir* directory. Missing directories are created automatically. This file is read in when each job is started, provided it exists, and is updated regularly with the job's current state.

`opt_logfile` **(default:** `log/optimizer-node_%04u.log`**)** Name mask for the log file used for possible error output originating from an optimizer. This mask contains a single `printf`-placeholder that is replaced by the number of the processing node that is running the optimizer. The file name is interpreted relative to the corresponding *job_dir*/*arg_dir* directory. Missing directories are created automatically. If a file with the given name already exists, alternative file names are tried until a free name is found; see `main_logfile` for details.

`sim_logfile` **(default:** `log/simulator-node_%04u.log`**)** Name mask for the log file used for possible error output originating from a simulator. This mask contains a single `printf`-placeholder that is replaced by the number of the processing node that is running the simulator. The file name is interpreted relative to the corresponding *job_dir*/*arg_dir* directory. Missing directories are created automatically. If a file with

the given name already exists, alternative file names are tried until a free name is found; see `main_logfile` for details.

`optimizer_map_file` **(default:** `dump/%012u-optmap.bin`**)** Name mask for the optimizer state file used for the DiCon-internal policy/`int` association map. This mask contains a single `printf`-placeholder that is replaced by the number of simulations done. The file name is interpreted relative to the corresponding *job_dir*`/`*arg_dir* directory. Missing directories are created automatically. If a file with the given name already exists, alternative file names are tried until a free name is found; see `main_logfile` for details.

`optimizer_lib_file` **(default:** `dump/%012u-optlib.bin`**)** Name mask for the optimizer state file used for the optimizer-internal state. This mask contains a single `printf`-placeholder that is replaced by the number of simulations done. The file name is interpreted relative to the corresponding *job_dir*`/`*arg_dir* directory. Missing directories are created automatically. If a file with the given name already exists, alternative file names are tried until a free name is found; see `main_logfile` for details.

`policy_bin_dumpfile` **(default:** `dump/%012u-policy.bin`**)** Name mask for the file used to dump the list of best policies in simulator-internal format. This mask contains a single `printf`-placeholder that is replaced by the number of simulations done. The file name is interpreted relative to the corresponding *job_dir*`/`*arg_dir* directory. Missing directories are created automatically. If a file with the given name already exists, alternative file names are tried until a free name is found; see `main_logfile` for details.

`policy_txt_dumpfile` **(default:** `dump/%012u-policy.txt`**)** Name mask for the file used to dump the list of best policies in human-readable format. This mask contains a single `printf`-placeholder that is replaced by the number of simulations done. The file name is interpreted relative to the corresponding *job_dir*`/`*arg_dir* directory. Missing directories are created automatically. If a file with the given name already exists, alternative file names are tried until a free name is found; see `main_logfile` for details.

`policy_count` **(default:** `10`**)** Maximum number of policies that are output in the files given by `policy_bin_dumpfile`, `policy_txt_dumpfile`, `policy_bin_result`, and `policy_txt_result`. An optimizer might not be able to return this many policies; in this case, it will return as many policies as it can.

`policy_bin_result` **(default:** `result-policy.bin`**)** Name of the file used to dump the list of best policies in simulator-internal format at the end of the optimization job. The file name is interpreted relative to the corresponding *job_dir*`/`*arg_dir* directory. Missing directories are created automatically. If a file with the given name already exists, alternative file names are tried until a free name is found; see `main_logfile` for details.

`policy_txt_result` **(default:** `result-policy.txt`**)** Name of the file used to dump the list of best policies in human-readable format at the end of the optimization job. The file name is interpreted relative to the corresponding *job_dir*`/`*arg_dir* directory. Missing directories are created automatically. If a file with the given name already exists, alternative file names are tried until a free name is found; see `main_logfile` for details.

A.2 `[job-...]` section

The following options define the actual optimization jobs and argument sets to perform during each DiCon run. There can be an arbitrary number of `[job-...]` sections with increasing numbers: `[job-1]`, `[job-2]`, `[job-3]`, etc. Each job section defines at least one argument set, given by the `opt_args[...]` and `sim_args[...]` parameters, using the argument specifier grammar described in section 3.4 on page 30.

`max_sims` **(default:** `0`**)** Maximum number of simulations to do in case the optimizer does not finish earlier. If this is 0, simulations are done until the optimization algorithm indicates that is has reached the end of the optimization job. Not all optimizers do so, so care must be taken that this does not lead to an infinite job.

`max_nodes` **(default:** `0`**)** Maximum number of simulator nodes to use in the cluster, in addition to the processing node that is running the optimizer. If this is 0, as many nodes as possible are used for simulation, depending on the number of available nodes in the cluster and the number of policies the optimizer can return at any given time. If this is 1, the simulator will be run on the same node as the optimizer, resulting in a combined optimizer/simulator node; this should be used for optimizers that can only return a single policy at a time, in order to avoid wasting resources.

`job_backlog` **(default:** `1`**)** Number of policies to request from the optimizer in addition to the ones that are currently being processed by simulators. DiCon will always try to get at least one more policy from the optimizer, so after all simulator backlogs have been filled and all simulators are busy, DiCon will request $job_backlog+1$ policies from the optimizer to hand out to simulators as soon as they finish a simulation.

`node_backlog` **(default:** `1`**)** Number of policies to send out to simulators in addition to the policy they are currently working on. Setting this option to a value different from 0 can improve performance in cases where simulations finish quickly or node intercommunication latency is high. Setting this value too high, though, might prevent additional simulators from being spawned since the optimizer in use might not be able to supply an arbitrarily large number of policies at once.

`checkpoint_secs` **(default:** `14400`**)** Maximum number of seconds between any two checkpoints. If this is 0, an arbitrary amount of time is allowed to pass before another checkpoint is enforced. The default value of 14,400 seconds is equivalent to 4 hours.

`checkpoint_sims` **(default:** `0`**)** Maximum number of simulations between any two checkpoints. If this is 0, an arbitrary number of simulations are allowed to complete before another checkpoint is enforced.

`checkpoints` **(no default)** Schedule specifier that describes a list of absolute numbers of simulations when to take a checkpoint; see the schedule specifier grammar described in section 3.4 on page 30 for details. If this is empty, checkpoints are not taken at predefined positions.

`optimizer` **(no default)** File name of the optimizer library to use. The file name is interpreted relative to the given working directory.

`simulator` **(no default)** File name of the simulator program to use. The file name is interpreted relative to the given working directory.

`opt_args[...]` **(no default)** List of positional arguments to the optimizer library. There can be an arbitrary number of `opt_args[...]` options with increasing numbers: `opt_args[1]`, `opt_args[2]`, `opt_args[3]`, etc. Each option describes a list of values for the corresponding positional argument, using the argument specifier grammar described in section 3.4 on page 30. If no `opt_args[...]` options are given, the optimizer is called with an empty set of command-line arguments. It is an error when one of the `opt_args[...]` options evaluates to an empty list.

`sim_args[...]` **(no default)** List of positional arguments to the simulator program. There can be an arbitrary number of `sim_args[...]` options with increasing numbers: `sim_args[1]`, `sim_args[2]`, `sim_args[3]`, etc. Each option describes a list of values for the corresponding positional argument, using the argument specifier grammar described in section 3.4 on page 30. If no `sim_args[...]` options are given, the simulator is called with an empty set of command-line arguments. It is an error when one of the `sim_args[...]` options evaluates to an empty list.

Appendix B

Source code

The following pages contain the entire source code of the Disease Control System DiCon, as of December 2009 (DiCon version `2009.12.0`). DiCon is free software: it can be redistributed and/or modified under the terms of the GNU General Public License [42] as published by the Free Software Foundation [43], either version 3 of the License, or any later version.

DiCon is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE; see the GNU General Public License in appendix C on page 228 for more details.

B.1 Miscellaneous

Listing B.1: README

```
Disease Control System DiCon
============================

Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
Designed and developed with the guidance of Nedialko B. Dimitrov
and Lauren Ancel Meyers at the University of Texas at Austin.


Introduction
------------

DiCon is a general optimization framework, with a special focus on
distributed, parallel disease simulation with policy optimization.


Distribution
------------

DiCon is free software: you can redistribute it and/or modify it under
the terms of the GNU General Public License as published by the Free
Software Foundation, either version 3 of the License, or (at your
option) any later version.

DiCon is distributed in the hope that it will be useful, but WITHOUT
ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
for more details.

You should have received a copy of the GNU General Public License
along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
```

Listing B.2: Makefile

```
gcc := g++
arch := native
program := dicon
mpi_path := /usr/include/mpi
boost_suffix := -mt  # can be some required suffix, such as '-mt', or blank
gcc_options := -Wall -pedantic -Os -march=$(arch) -I$(mpi_path) -DDICON_PROGRAM_NAME=\"$(program)\"
libraries := -lboost_filesystem$(boost_suffix) -lboost_mpi$(boost_suffix) -lboost_regex$(boost_suffix) -ldl -lgmpxx -lprotobuf
debug := yes  # 'yes' adds debugging information and assertions to the program, use 'no' to deactivate.

#-------------------------------------------------------------------------------
.PHONY: all python simulator optimizer clean

ifeq "yes" "$(strip $(debug))"
gcc_options := -g -UNDEBUG $(gcc_options)
else
gcc_options := -DNDEBUG $(gcc_options)
endif

dicon_proto := simulator
simulator_proto := simulator

dicon_libraries := $(libraries)
simulator_libraries := -lboost_serialization$(boost_suffix) -lprotobuf
optimizer_libraries :=

define copy-file
@echo "[INSTALL]    $@"
@rm -f $@
@cp $< $@
endef

# Make everything.
all: dicon python simulator optimizer

# Build DiCon executable.
dicon: $(patsubst %.cpp,%.o,$(wildcard src/*.cpp)) $(patsubst %,protobuf/cpp/%.pb.o,$(dicon_proto))
        @echo "[BUILD]      $@"
        @$(gcc) $(gcc_options) $(dicon_libraries) -o $@ $^

# Compile C++ DiCon files.
$(patsubst %.cpp,%.o,$(wildcard src/*.cpp)): \
src/%.o: src/%.cpp
        @echo "[COMPILE]    $(patsubst src/%.cpp,src/%.o,$<)"
        @$(gcc) $(gcc_options) -c -o $(patsubst src/%.cpp,src/%.o,$<) $<

# Compile C++ protobuf files.
$(patsubst protobuf/%.proto,protobuf/cpp/%.pb.o,$(wildcard protobuf/*.proto)): \
protobuf/cpp/%.pb.o: protobuf/cpp/%.pb.cc protobuf/cpp/%.pb.h
        @echo "[COMPILE]    $@"
        @$(gcc) $(gcc_options) -c -o $@ $<

# Create C++ files from protobuf.
define run-cpp-protobuf
@echo "[PROTOBUF]   $(patsubst protobuf/%.proto,protobuf/cpp/%.pb.cc (and .h),$<)"
@( cd protobuf && mkdir -p cpp && protoc --cpp_out cpp $(patsubst protobuf/%,%,$<) )
endef
$(patsubst protobuf/%.proto,protobuf/cpp/%.pb.cc,$(wildcard protobuf/*.proto)): \
protobuf/cpp/%.pb.cc: protobuf/%.proto
        $(run-cpp-protobuf)
$(patsubst protobuf/%.proto,protobuf/cpp/%.pb.h,$(wildcard protobuf/*.proto)): \
protobuf/cpp/%.pb.h: protobuf/%.proto
        $(run-cpp-protobuf)
$(patsubst %,simulator/cpp/%.pb.cc,$(simulator_proto)): \
simulator/cpp/%.pb.cc: protobuf/cpp/%.pb.cc
        $(copy-file)
$(patsubst %,simulator/cpp/%.pb.h,$(simulator_proto)): \
```

```
simulator/cpp/%.pb.h: protobuf/cpp/%.pb.h
	$(copy-file)

# Create Python files from protobuf.
$(patsubst protobuf/%.proto,protobuf/python/%_pb2.py,$(wildcard protobuf/*.proto)): \
protobuf/python/%_pb2.py: protobuf/%.proto
	@echo "[PROTOBUF]  $(patsubst protobuf/%.proto,protobuf/python/%_pb2.py,$<)"
	@( cd protobuf && mkdir -p python && protoc --python_out python $(patsubst protobuf/%,%,$<) )
$(patsubst %,simulator/python/%_pb2.py,$(simulator_proto)): \
simulator/python/%_pb2.py: protobuf/python/%_pb2.py
	$(copy-file)

# Compile C++ simulators.
simulator: $(patsubst simulator/cpp/%/main.cpp,simulator/%,$(wildcard simulator/cpp/*/main.cpp))
simulator_targets :=
define build-simulator
simulator/$(1): $(wildcard simulator/cpp/$(1)/*.*pp) $(wildcard simulator/cpp/*.*pp) $(patsubst %,simulator/cpp/%.pb.h,$(simulator_proto))
simulator/$(1): $(wildcard simulator/cpp/$(1)/*.cpp) $(wildcard simulator/cpp/*.cpp) $(patsubst %,simulator/cpp/%.pb.cc,$(simulator_proto))
	@echo "[BUILD]     $$@"
	@$(gcc) $(gcc_options) $(simulator_libraries) -o $$@ $(wildcard simulator/cpp/$(1)/*.cpp) $(wildcard simulator/cpp/*.cpp) $(patsubst %,simulator/cpp/%.pb.cc,$(simulator_proto))
simulator_targets += simulator/$(1)
endef
$(foreach simulator,$(patsubst simulator/cpp/%/main.cpp,%,$(wildcard simulator/cpp/*/main.cpp)),$(eval $(call build-simulator,$(simulator))))

# Compile C++ optimizers.
optimizer: $(patsubst optimizer/cpp/%/main.cpp,optimizer/%.so,$(wildcard optimizer/cpp/*/main.cpp))
optimizer_targets :=
define build-optimizer
optimizer/$(1).so: $(wildcard optimizer/cpp/$(1)/*.*pp) $(wildcard optimizer/cpp/*.*pp)
optimizer/$(1).so: $(wildcard optimizer/cpp/$(1)/*.cpp) $(wildcard optimizer/cpp/*.cpp)
	@echo "[BUILD]     $$@"
	@$(gcc) $(gcc_options) $(optimizer_libraries) -fPIC -shared -o $$@ $(wildcard optimizer/cpp/$(1)/*.cpp) $(wildcard optimizer/cpp/*.cpp)
optimizer_targets += optimizer/$(1).so
endef
$(foreach optimizer,$(patsubst optimizer/cpp/%/main.cpp,%,$(wildcard optimizer/cpp/*/main.cpp)),$(eval $(call build-optimizer,$(optimizer))))

# Prepare Python files.
python: $(patsubst %,simulator/python/%_pb2.py,$(simulator_proto))

# Make dependencies for C++ DiCon files.
$(patsubst %.cpp,%.d,$(wildcard src/*.cpp)): \
src/%.d: src/%.cpp
	@( cd src && $(gcc) $(gcc_options) -MM -MG -MP -MT "$(patsubst src/%.cpp,%.o,$<) $(patsubst src/%.cpp,%.d,$<)" -MF $(patsubst src/%.cpp,%.d_tmp,$<) $(patsubst src/%.cpp,%.cpp,$<) && sed -e 's#\([a-zA-Z0-9./_]\+\)#src/\1#g' -e 's#[a-zA-Z0-9._]\+/\.\./##g' $(patsubst src/%.cpp,%.d_tmp,$<) > $(patsubst src/%.cpp,%.d,$<) && rm -f $(patsubst src/%.cpp,%.d_tmp,$<) )

# Include dependency files created for C++ DiCon files.
-include $(patsubst src/%.cpp,src/%.d,$(wildcard src/*.cpp))

# Clean up all generated files.
clean:
	@echo "[CLEAN]"
	@rm -f \
dicon \
$(patsubst %.cpp,%.o,$(wildcard src/*.cpp)) \
$(patsubst %.cpp,%.d,$(wildcard src/*.cpp)) \
$(patsubst protobuf/%.proto,protobuf/cpp/%.pb.o,$(wildcard protobuf/*.proto)) \
$(patsubst protobuf/%.proto,protobuf/cpp/%.pb.cc,$(wildcard protobuf/*.proto)) \
$(patsubst protobuf/%.proto,protobuf/cpp/%.pb.h,$(wildcard protobuf/*.proto)) \
$(patsubst protobuf/%.proto,protobuf/python/%_pb2.py,$(wildcard protobuf/*.proto)) \
$(patsubst %,simulator/cpp/%.pb.cc,$(simulator_proto)) \
$(patsubst %,simulator/cpp/%.pb.h,$(simulator_proto)) \
$(patsubst %,simulator/python/%_pb2.py,$(simulator_proto)) \
$(simulator_targets) \
$(optimizer_targets)
```

Listing B.3: demo/main.conf

```
; Sample DiCon configuration file.
;
; Leading and trailing whitespace is ignored, as are empty lines and
; lines starting with ';' or '#'. Section names and parameter names
; are case-sensitive, leading and trailing whitespace in names and
; values is also ignored.

[global]
;main_logfile=log/main.log
;main_log_level=info
;node_logfile=log/node_%04u.log
;node_log_level=info
;max_jobs=1000000
;job_dir=job-%u
;arg_dir=argument-%05u
;job_logfile=job.log
;job_log_level=info
;checkpoint=checkpoint.xml
;opt_logfile=log/optimizer-node_%04u.log
;sim_logfile=log/simulator-node_%04u.log
;optimizer_map_file=dump/%012u-optmap.bin
;optimizer_lib_file=dump/%012u-optlib.bin
;policy_bin_dumpfile=dump/%012u-policy.bin
;policy_txt_dumpfile=dump/%012u-policy.txt
;policy_count=10
;policy_bin_result=result-policy.bin
```

```
;policy_txt_result=result-policy.txt

main_log_level=debug
node_log_level=debug
job_log_level=debug

[job-1]
;max_sims=0
;max_nodes=0
;job_backlog=1
;node_backlog=1
;checkpoint_secs=14400
;checkpoint_sims=0
;checkpoints=
;optimizer=
;simulator=
;opt_args[1]=
;sim_args[1]=

max_nodes=0
job_backlog=1
node_backlog=1
checkpoint_secs=0
checkpoint_sims=0
optimizer=../optimizer/exhaustive.so
simulator=../simulator/simple_simulator.py

[job-2]
; Some other job set here.
```

Listing B.4: protobuf/simulator.proto

```
package proto.simulator;


enum Type {
  CHILDREN = 1;
  SIMULATE = 2;
  DISPLAY = 3;
}


message QuestionChildren {
  repeated bytes node = 1;
}

message QuestionSimulate {
  repeated bytes node = 1;
}

message QuestionDisplay {
  repeated bytes node = 1;
}

message Question {
  required Type type = 99;
  optional QuestionChildren q_children = 1;
  optional QuestionSimulate q_simulate = 2;
  optional QuestionDisplay q_display = 3;
}


message AnswerChildren {
  repeated bytes node = 1;
}

message AnswerSimulate {
  required double reward = 1;
}

message AnswerDisplay {
  required string display = 1;
}

message AnswerFailure {
  required string what = 1;
}

message Answer {
  required bool failed = 99;
  optional AnswerFailure a_failure = 98;
  optional AnswerChildren a_children = 1;
  optional AnswerSimulate a_simulate = 2;
  optional AnswerDisplay a_display = 3;
}
```

B.2 Main system

Listing B.5: src/communicator.hpp

```
1  #ifndef DICON_COMMUNICATOR_HPP_
2  #define DICON_COMMUNICATOR_HPP_
3
4  /*---- [Distribution] ------------------------------------------------
5   * This file is part of the Disease Control System DiCon.
6   *
7   * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8   * Designed and developed with the guidance of Nedialko B. Dimitrov
9   * and Lauren Ancel Meyers at the University of Texas at Austin.
10  *
11  * DiCon is free software: you can redistribute it and/or modify it
12  * under the terms of the GNU General Public License as published by
13  * the Free Software Foundation, either version 3 of the License, or
14  * (at your option) any later version.
15  *
16  * DiCon is distributed in the hope that it will be useful, but
17  * WITHOUT ANY WARRANTY; without even the implied warranty of
18  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19  * General Public License for more details.
20  *
21  * You should have received a copy of the GNU General Public License
22  * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23  *--------------------------------------------------------------------*/
24
25 /**
26  * @file
27  * @brief Communicator class.
28  */
29 #include <boost/noncopyable.hpp>
30 #include <string>
31
32
33 /**
34  * @brief Message-based communication.
35  *
36  * The Communicator class provides a %message-based communication
37  * channel.  This channel can be created on top of arbitrary file
38  * descriptors, e.g., standard input and output, files in the
39  * filesystem, network sockets, etc.  Each %message sent over the
40  * communication channel is encoded by first giving the length of the
41  * %message, followed by the %message itself. The length is encoded
42  * using little-endian encoding of a 32-bit unsigned integer within 4
43  * bytes. The %message is sent as-is, using exactly the number of
44  * bytes given by the encoded %message length.  For example, the
45  * %message <code>Hello, World!</code> (13 characters) would be
46  * transmitted as follows.
47  *
48  * @verbatim
49  |  0x0d 0x00 0x00 0x00  |  0x48 0x65 0x6c 0x6c 0x6f 0x2c 0x20 0x57 0x
    x6f 0x72 0x6c 0x64 0x21  |
50  |    13    0    0    0  |    72  101  108  108  111   44   32   87
    111  114  108  100   33  |
51  |                  13  |   'H'  'e'  'l'  'l'  'o'  ','  ' '  'W'  'o
    '  'r'  'l'  'd'  '!'  |
52  @endverbatim
53  *
54  * As the length of each %message is given explicitly, there is no
55  * need for any special separator character.  Therefore, multiple
56  * messages are sent immediately one after the other, with no extra
57  * padding in-between.
58  */
59 class Communicator
60   : boost::noncopyable
61 {
62 public:
63   /**
64    * @brief Create communicator on top of pair of file descriptors.
65    *
66    * Constructor that creates a new communicator with the given input
67    * and output file descriptors. The communicator will read from file
68    * descriptor @e input and write to file descriptor @e output. After
69    * creation, the communicator owns the file descriptors and closes
70    * them when shutting down.
71    *
72    * @param input Input file descriptor.
73    * @param output Output file descriptor.
74    */
75   Communicator( int input, int output );
76   /**
77    * @brief Shutdown communicator.
78    *
79    * Destructor that shuts down the communicator. This also closes
80    * all associated file descriptors.
81    */
82   ~Communicator();
83
84 public:
85   /**
86    * @brief Receive %message.
87    *
88    * Receive the next %message from the communication channel. This
89    * call blocks until an entire %message is received.
90    *
91    * @return Received %message.
92    * @throws IOReadError when not enough data could be read.
93    */
94   std::string receive_message();
95   /**
96    * @brief Send %message.
97    *
98    * Send a %message to the communication channel. This call blocks
99    * until the entire %message given by @e %message is sent. After
100   * sending the %message this call also flushes the associated output
101   * file descriptor.
102   *
103   * @param message Message to be sent.
104   * @throws IOWriteError when not all data could be written.
105   */
106  void send_message( const std::string &message );
107
108 private:
109  std::string read( size_t count );
```

```
110     void write( const std::string &buffer );
111     void flush();
112
113   private:
114     int input_;
115     int output_;
116   };
117
118   #endif //DICON_COMMUNICATOR_HPP_
```

Listing B.6: src/communicator.cpp

```
1   /*--- [Distribution] ------------------------------------------------------
2    * This file is part of the Disease Control System DiCon.
3    *
4    * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
5    * Designed and developed with the guidance of Nedialko B. Dimitrov
6    * and Lauren Ancel Meyers at the University of Texas at Austin.
7    *
8    * DiCon is free software: you can redistribute it and/or modify it
9    * under the terms of the GNU General Public License as published by
10   * the Free Software Foundation, either version 3 of the License, or
11   * (at your option) any later version.
12   *
13   * DiCon is distributed in the hope that it will be useful, but
14   * WITHOUT ANY WARRANTY; without even the implied warranty of
15   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
16   * General Public License for more details.
17   *
18   * You should have received a copy of the GNU General Public License
19   * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
20   *-----------------------------------------------------------------------*/
21
22  #include "communicator.hpp"
23
24  #include "error.hpp"
25  #include "file.hpp"
26  #include <cerrno>
27  #include <boost/scoped_array.hpp>
28
29
30  namespace detail {
31
32    static
33    std::string
34    encode_uint32( uint32_t value ) {
35      std::string message( 4, char(0) );
36
37      message[0] = char(static_cast<unsigned char>((value >>  0) & 255));
38      message[1] = char(static_cast<unsigned char>((value >>  8) & 255));
39      message[2] = char(static_cast<unsigned char>((value >> 16) & 255));
40      message[3] = char(static_cast<unsigned char>((value >> 24) & 255));
41
42      return message;
43    }
44
45
46    static
47    uint32_t
48    decode_uint32( const std::string &buffer ) {
49      if( buffer.size() != 4 ) {
50        BOOST_THROW_EXCEPTION( AssertionError()
51                               << errinfo_value<size_t>(buffer.size())
52                               << errinfo_expected_value<size_t>(4) );
53      }
54
55      return
56        (uint32_t(static_cast<unsigned char>(buffer[0])) <<  0) |
57        (uint32_t(static_cast<unsigned char>(buffer[1])) <<  8) |
58        (uint32_t(static_cast<unsigned char>(buffer[2])) << 16) |
59        (uint32_t(static_cast<unsigned char>(buffer[3])) << 24);
60    }
61
62  }
63
64
65
66  Communicator::Communicator( int input, int output )
67    : input_(input), output_(output)
68  {
69  }
70
71
72  Communicator::~Communicator() {
73    close( input_ );
74    close( output_ );
75    // Ignore errors.
76  }
77
78
79  std::string
80  Communicator::receive_message() {
81    return read( detail::decode_uint32(read(4)) );
82  }
83
84
85  void
86  Communicator::send_message( const std::string &message ) {
87    write( detail::encode_uint32(message.length()) );
88    write( message );
89    flush();
90  }
91
92
93  std::string
94  Communicator::read( size_t count ) {
95    boost::scoped_array<char> buffer( new char[count] );
96
97    size_t done = 0;
98    while( done < count ) {
99      ssize_t res;
100     if( (res = ::read(input_, buffer.get()+done, count-done)) <= 0 ) {
101       BOOST_THROW_EXCEPTION( IOReadError() << errinfo_api_function("read")
102                              << errinfo_errno(errno) << errinfo_file_descriptor(input_) );
```

```
103       }
104
105       assert( size_t(res) <= count-done );
106
107       done += size_t(res);
108     }
109
110     return std::string( buffer.get(), count );
111   }
112
113
114   void
115   Communicator::write( const std::string &buffer ) {
116     const size_t count = buffer.size();
117
118     size_t done = 0;
119     while( done < count ) {
120       ssize_t res;
121       if( (res = ::write(output_, buffer.c_str()+done, count-done)) <= 0 ) {
122         BOOST_THROW_EXCEPTION( IOWriteError() << errinfo_api_function("write")
123                                << errinfo_errno(errno) << errinfo_file_descriptor(output_) );
124       }
125
126       assert( size_t(res) <= count-done );
127
128       done += size_t(res);
129     }
130   }
131
132
133   void
134   Communicator::flush() {
135     if( ::fsync(output_) != 0 && errno != EINVAL ) {
136       BOOST_THROW_EXCEPTION( IOError() << errinfo_api_function("fsync")
137                              << errinfo_errno(errno) << errinfo_file_descriptor(output_) );
138     }
139   }
```

Listing B.7: src/config.hpp

```
1   #ifndef DICON_CONFIG_HPP_
2   #define DICON_CONFIG_HPP_
3
4   /*--- [Distribution] ---------------------------------------------------
5    * This file is part of the Disease Control System DiCon.
6    *
7    * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8    * Designed and developed with the guidance of Nedialko B. Dimitrov
9    * and Lauren Ancel Meyers at the University of Texas at Austin.
10   *
11   * DiCon is free software: you can redistribute it and/or modify it
12   * under the terms of the GNU General Public License as published by
13   * the Free Software Foundation, either version 3 of the License, or
14   * (at your option) any later version.
15   *
16   * DiCon is distributed in the hope that it will be useful, but
17   * WITHOUT ANY WARRANTY; without even the implied warranty of
18   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19   * General Public License for more details.
20   *
21   * You should have received a copy of the GNU General Public License
22   * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23   *-----------------------------------------------------------------------*/
24
25  /**
26   * @file
27   * @brief Configuration class.
28   */
29  #include "error.hpp"
30  #include <map>
31  #include <string>
32
33
34  /// Errinfo storing the config filename.
35  DICON_ERRINFO( config_file      , std::string );
36  /// Errinfo storing the line number in the config file.
37  DICON_ERRINFO( config_line      , int         );
38  /// Errinfo storing the section name in the config file.
39  DICON_ERRINFO( config_section   , std::string );
40  /// Errinfo storing the parameter name in the config file.
41  DICON_ERRINFO( config_parameter, std::string );
42  /// Errinfo storing the parameter value in the config file.
43  DICON_ERRINFO( config_value     , std::string );
44
45
46  /// Config file related error.
47  struct ConfigError : virtual Error
48  { virtual const char *what() const throw() { return "Config_file_related_error."; } };
49
50  /// Config file has invalid format.
51  struct ConfigFormatError : virtual ConfigError
52  { virtual const char *what() const throw() { return "Config_file_has_invalid_format."; } };
53
54  /// Config parameter has invalid format.
55  struct ConfigInvalidParameterError : virtual ConfigError
56  { virtual const char *what() const throw() { return "Config_parameter_has_invalid_format."; } };
57
58
59  /// Config item not found.
60  struct ConfigNotFoundError : virtual ConfigError
61  { virtual const char *what() const throw() { return "Config_item_not_found."; } };
62
63  /// Config section not found in file.
64  struct ConfigSectionNotFoundError : virtual ConfigNotFoundError
65  { virtual const char *what() const throw() { return "Config_section_not_found_in_file."; } };
66
67  /// Config parameter not found in section.
68  struct ConfigParameterNotFoundError : virtual ConfigNotFoundError
69  { virtual const char *what() const throw() { return "Config_parameter_not_found_in_section."; } };
70
```

```
71
72  /// Config item has invalid name.
73  struct ConfigInvalidNameError : virtual ConfigFormatError
74  { virtual const char *what() const throw() { return "Config item has
    invalid name."; } };
75
76  /// Config section has invalid name.
77  struct ConfigInvalidSectionNameError : virtual ConfigInvalidNameError
78  { virtual const char *what() const throw() { return "Config section
    has invalid name."; } };
79
80  /// Config parameter has invalid name.
81  struct ConfigInvalidParameterNameError : virtual
    ConfigInvalidNameError
82  { virtual const char *what() const throw() { return "Config parameter
    has invalid name."; } };
83
84
85  /// Config item already defined.
86  struct ConfigRedefinedError : virtual ConfigFormatError
87  { virtual const char *what() const throw() { return "Config item
    already defined."; } };
88
89  /// Config section already defined in file.
90  struct ConfigSectionRedefinedError : virtual ConfigRedefinedError
91  { virtual const char *what() const throw() { return "Config section
    already defined in file."; } };
92
93  /// Config parameter already defined in section.
94  struct ConfigParameterRedefinedError : virtual ConfigRedefinedError
95  { virtual const char *what() const throw() { return "Config parameter
    already defined in section."; } };
96
97
98  /// Unexpected config item.
99  struct ConfigLoneError : virtual ConfigFormatError
100 { virtual const char *what() const throw() { return "Unexpected config
    item."; } };
101
102 /// Unexpected line format in config file.
103 struct ConfigLoneLineError : virtual ConfigLoneError
104 { virtual const char *what() const throw() { return "Unexpected line
    format in config file."; } };
105
106 /// Unexpected parameter definition in config file.
107 struct ConfigLoneParameterError : virtual ConfigLoneError
108 { virtual const char *what() const throw() { return "Unexpected
    parameter definition in config file."; } };
109
110
111 /// Config file could not be imported.
112 struct ConfigImportError : public virtual ConfigError, public virtual
    IOError
113 { virtual const char *what() const throw() { return "Config file could
    not be imported."; } };
114
115
116 /**
117  * @brief Name-value based configuration.
118  *
119  * The Configuration  class provides access  to configuration settings
120  * based on name-value pairs  arranged in sections. Configurations can
121  * be read from configuration files in the INI file format.
122  */
123 class Configuration {
124 public:
125   /**
126    * @brief Check if section exists.
127    *
128    * Check  if  the  section  given  by  @e  section  exists  in  the
129    * configuration. A section  is said to exist if and  only if it has
130    * at least one parameter set.
131    *
132    * @param section Section name.
133    * @returns @c true iff the section exists.
134    */
135   bool has( const std::string &section ) const;
136   /**
137    * @brief Check if parameter exists in section.
138    *
139    * Check if  the parameter  given by @e  name exists in  the section
140    * given by @e section. A parameter in a section is said to exist if
141    * and only if it is set.
142    *
143    * @param section Section name.
144    * @param name Parameter name within the section.
145    * @returns @c true iff the parameter exists in the section.
146    */
147   bool has( const std::string &section, const std::string &name )
    const;
148
149   /**
150    * @brief Get value of parameter from section.
151    *
152    * Get the  value of the parameter  given by @e name  in the section
153    * given by  @e section.  If this parameter  is not set,  this call
154    * fails. The value is converted to  the type given by @e T using @c
155    * boost::lexical_cast<>.
156    *
157    * @tparam T Value type.
158    * @param section Section name.
159    * @param name Parameter name within the section.
160    * @returns Value of the parameter.
161    * @throws  ConfigSectionNotFoundError  when  the section  does  not
162    *    exist.
163    * @throws ConfigParameterNotFoundError when  the section exists but
164    *    the parameter does not exist in the section.
165    * @throws ConfigInvalidParameterError when  the parameter exists in
166    *    the section but  its value could not be  converted to the given
167    *    type.
168    */
169   template< typename T >
170   T get( const std::string &section, const std::string &name ) const;
171   /**
172    * @brief Get value of parameter from section with fallback.
173    *
174    * Get the  value of the parameter  given by @e name  in the section
175    * given by  @e section.  If this parameter  is not set,  the value
176    * given by @e fallback is  returned.  The value is converted to the
177    * type given by @e T using @c boost::lexical_cast<>.
178    *
179    * @tparam T Value type.
```

```
180     * @param section Section name.
181     * @param name Parameter name within the section.
182     * @param fallback Fallback value to return when parameter does not
183     *   exist.
184     * @returns Value of the parameter.
185     * @throws ConfigInvalidParameterError when the parameter exists in
186     *   the section but its value could not be converted to the given
187     *   type.
188     */
189    template< typename T >
190    T get( const std::string &section, const std::string &name, const T
    &fallback ) const;
191
192  public:
193    /**
194     * @brief Put value of parameter in section.
195     *
196     * Put the value given by @e value in the parameter given by @e name
197     * in the section given by @e section. This call fails if the value
198     * of type given by @e T can not be converted to the internal
199     * storage type using @c boost::lexical_cast<>.
200     *
201     * @tparam T Value type.
202     * @param section Section name.
203     * @param name Parameter name within the section.
204     * @param value New value of the parameter.
205     * @throws ConfigInvalidParameterError when the value can not be
206     *   converted to the internal storage type.
207     */
208    template< typename T >
209    void put( const std::string &section, const std::string &name, const
    T &value );
210
211  public:
212    /**
213     * @brief Clear configuration.
214     *
215     * Clear the configuration. This removes all parameters, so that
216     * after this call returns no section and no parameter exists within
217     * the configuration.
218     */
219    void clear();
220    /**
221     * @brief Import configuration from stream.
222     *
223     * Import a configuration in the INI file format from the given
224     * stream. This merges the current configuration with the
225     * configuration read from the stream, so that parameters from the
226     * stream replace parameters already existing within the
227     * configuration.
228     *
229     * @param input Input stream.
230     * @throws ConfigInvalidSectionNameError when an invalid section
231     *   name was found.
232     * @throws ConfigInvalidParameterNameError when an invalid parameter
233     *   name was found.
234     * @throws ConfigSectionRedefinedError when a section was defined
235     *   more than once within the INI file.
236     * @throws ConfigParameterRedefinedError when a parameter was
237     *   defined more than once within the same section within the INI
238     *   file.
239     * @throws ConfigLoneParameterError when a parameter was defined
240     *   without a section.
241     * @throws ConfigLoneLineError when a line that is neither section
242     *   nor parameter definition was found.
243     */
244    void import( std::istream &input );
245    /**
246     * @brief Import configuration from file.
247     *
248     * Import a configuration in the INI file format from the given
249     * file. This merges the current configuration with the
250     * configuration read from the stream, so that parameters from the
251     * stream replace parameters already existing within the
252     * configuration.
253     *
254     * @param filename Input filename.
255     * @throws ConfigInvalidSectionNameError when an invalid section
256     *   name was found.
257     * @throws ConfigInvalidParameterNameError when an invalid parameter
258     *   name was found.
259     * @throws ConfigSectionRedefinedError when a section was defined
260     *   more than once within the INI file.
261     * @throws ConfigParameterRedefinedError when a parameter was
262     *   defined more than once within the same section within the INI
263     *   file.
264     * @throws ConfigLoneParameterError when a parameter was defined
265     *   without a section.
266     * @throws ConfigLoneLineError when a line that is neither section
267     *   nor parameter definition was found.
268     * @throws ConfigImportError when an error occurred while reading
269     *   the input file.
270     */
271    void import( const std::string &filename );
272
273  private:
274    template< typename T >
275    T get_( const std::string &section, const std::string &name, const T
    *fallback ) const;
276
277  private:
278    typedef std::map< std::string
279                    , std::map<std::string, std::string>
280                    > values_t;
281    values_t values_;
282  };
283
284
285  #include "config.ipp"
286
287  #endif //DICON_CONFIG_HPP_
```

Listing B.8: src/config.ipp

```
1  // -*- c++ -*-
2
3  /*---- [Distribution] ------------------------------------------------
4   * This file is part of the Disease Control System DiCon.
5   *
6   * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
```

```
 7  * Designed and developed with the guidance of Nedialko B. Dimitrov
 8  * and Lauren Ancel Meyers at the University of Texas at Austin.
 9  *
10  * DiCon is free software: you can redistribute it and/or modify it
11  * under the terms of the GNU General Public License as published by
12  * the Free Software Foundation, either version 3 of the License, or
13  * (at your option) any later version.
14  *
15  * DiCon is distributed in the hope that it will be useful, but
16  * WITHOUT ANY WARRANTY; without even the implied warranty of
17  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
18  * General Public License for more details.
19  *
20  * You should have received a copy of the GNU General Public License
21  * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
22  *-----------------------------------------------------------------*/
23
24 #include <boost/format.hpp>
25 #include <boost/lexical_cast.hpp>
26
27
28 template< typename T >
29 inline
30 T
31 Configuration::get_( const std::string &section, const std::string &
    name, const T *fallback ) const {
32   try {
33     values_t::const_iterator it;
34     if( (it = values_.find(section)) == values_.end() ) {
35       if( fallback )
36         return *fallback;
37       else
38         BOOST_THROW_EXCEPTION( ConfigSectionNotFoundError() );
39     }
40
41     std::map<std::string, std::string>::const_iterator jt;
42     if( (jt = it->second.find(name)) == it->second.end() ) {
43       if( fallback )
44         return *fallback;
45       else
46         BOOST_THROW_EXCEPTION( ConfigParameterNotFoundError() );
47     }
48
49     try {
50       return boost::lexical_cast<T>( jt->second );
51     }
52     catch( boost::bad_lexical_cast &e ) {
53       BOOST_THROW_EXCEPTION( ConfigInvalidParameterError()
54                              << errinfo_config_value(jt->second)
55                              << errinfo_nested_exception(boost::
    copy_exception(e)) );
56     }
57   }
58   catch( boost::exception &e ) {
59     e << errinfo_config_section(section)
60       << errinfo_config_parameter(name);
61     throw;
62   }
63 }
64
65
```

```
66 template< typename T >
67 inline
68 T
69 Configuration::get( const std::string &section, const std::string &
    name ) const {
70   return get_<T>( section, name, NULL );
71 }
72
73
74 template< typename T >
75 inline
76 T
77 Configuration::get( const std::string &section, const std::string &
    name, const T &fallback ) const {
78   return get_<T>( section, name, &fallback );
79 }
80
81
82 template< typename T >
83 inline
84 void
85 Configuration::put( const std::string &section, const std::string &
    name, const T &value ) {
86   try {
87     values_[section][name] = boost::lexical_cast<std::string>( value )
    ;
88   }
89   catch( boost::bad_lexical_cast &e ) {
90     BOOST_THROW_EXCEPTION( ConfigInvalidParameterError()
91                            << errinfo_config_section(section)
92                            << errinfo_config_parameter(name)
93                            << errinfo_config_value(value)
94                            << errinfo_nested_exception(boost::
    copy_exception(e)) );
95   }
96 }
```

Listing B.9: src/config.cpp

```
 1 /*---- [Distribution] ------------------------------------------------
 2  * This file is part of the Disease Control System DiCon.
 3  *
 4  * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
 5  * Designed and developed with the guidance of Nedialko B. Dimitrov
 6  * and Lauren Ancel Meyers at the University of Texas at Austin.
 7  *
 8  * DiCon is free software: you can redistribute it and/or modify it
 9  * under the terms of the GNU General Public License as published by
10  * the Free Software Foundation, either version 3 of the License, or
11  * (at your option) any later version.
12  *
13  * DiCon is distributed in the hope that it will be useful, but
14  * WITHOUT ANY WARRANTY; without even the implied warranty of
15  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
16  * General Public License for more details.
17  *
18  * You should have received a copy of the GNU General Public License
19  * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
20  *-----------------------------------------------------------------*/
```

```
21
22 #include "config.hpp"
23 #include <boost/foreach.hpp>
24 #include <boost/format.hpp>
25 #include <boost/optional.hpp>
26 #include <cassert>
27 #include <fstream>
28 #include <iostream>
29 #include <set>
30 #include <stdexcept>
31
32
33 namespace detail {
34
35   static const char config_whitespace[] = " \n\r\t";
36
37   static const char config_comment[] = ";#";
38
39   static const char config_section_open  = '[';
40   static const char config_section_close = ']';
41
42   static const char config_separator = '=';
43
44   static const char config_name[] =
45     "abcdefghijklmnopqrstuvwxyz"
46     "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
47     "0123456789"
48     "[]-_"
49     ;
50
51
52   static
53   std::string
54   strip( const std::string &text ) {
55     std::string::size_type first;
56     if( (first = text.find_first_not_of(config_whitespace)) == std::
   string::npos ) {
57       // Found no non-blank character.
58       return std::string();
59     }
60
61     std::string::size_type last = text.find_last_not_of(
   config_whitespace);
62     assert( last != std::string::npos );
63     assert( first <= last );
64
65     return text.substr( first, last-first+1 );
66   }
67
68
69   static
70   bool
71   is_blank( const std::string &line ) {
72     return line.empty();
73   }
74
75
76   static
77   bool
78   is_comment( const std::string &line ) {
79     return !line.empty() && std::string(config_comment).find(line[0])
   != std::string::npos;
80   }
81
82
83   static
84   bool
85   is_section( const std::string &line, std::string &section ) {
86     if( !line.empty() && line[0] == config_section_open && line[line.
   size()-1] == config_section_close ) {
87       section = strip( line.substr(1, line.size()-2) );
88       return true;
89     }
90
91     return false;
92   }
93
94
95   static
96   bool
97   is_parameter( const std::string &line, std::string &name, std::
   string &value ) {
98     std::string::size_type pos;
99     if( (pos = line.find(config_separator)) != std::string::npos ) {
100      name = strip( line.substr(0, pos) );
101      value = strip( line.substr(pos+1) );
102      return true;
103    }
104
105    return false;
106  }
107
108
109  static
110  bool
111  is_valid_name( const std::string &name ) {
112    return name.find_first_not_of(config_name) == std::string::npos;
113  }
114
115
116  static
117  bool
118  is_valid_section_name( const std::string &section ) {
119    return is_valid_name( section );
120  }
121
122
123  static
124  bool
125  is_valid_parameter_name( const std::string &name ) {
126    return is_valid_name( name );
127  }
128
129 }
130
131
132
133 void
134 Configuration::clear() {
135   values_.clear();
136 }
```

```
137
138
139 bool
140 Configuration::has( const std::string &section ) const {
141   if( values_.find(section) == values_.end() )
142     return false;
143
144   return true;
145 }
146
147
148 bool
149 Configuration::has( const std::string &section, const std::string &name ) const {
150   values_t::const_iterator it;
151
152   if( (it = values_.find(section)) == values_.end() )
153     return false;
154
155   if( it->second.find(name) == it->second.end() )
156     return false;
157
158   return true;
159 }
160
161
162 void
163 Configuration::import( std::istream &input ) {
164   boost::optional<std::string> current_section;
165
166   unsigned lineno;
167   std::string line;
168
169   std::map<std::string, std::set<std::string> > parameters;
170
171   try {
172     for( lineno = 1; std::getline(input, line); ++lineno ) {
173       // Remove leading and trailing whitespace.
174       line = detail::strip( line );
175
176       if( detail::is_blank(line) || detail::is_comment(line) ) {
177         // Ignore comments and blank lines.
178         continue;
179       }
180
181       std::string section;
182       std::string name, value;
183
184       if( detail::is_section(line, section) ) {
185         if( !detail::is_valid_section_name(section) )
186           BOOST_THROW_EXCEPTION( ConfigInvalidSectionNameError() << errinfo_config_section(section) );
187
188         if( parameters.find(section) != parameters.end() )
189           BOOST_THROW_EXCEPTION( ConfigSectionRedefinedError() << errinfo_config_section(section) );
190
191         // Section "marker".
192         parameters[section];
193
194         current_section = section;
195       }
196       else if( detail::is_parameter(line, name, value) ) {
197         if( !current_section )
198           BOOST_THROW_EXCEPTION( ConfigLoneParameterError() << errinfo_config_parameter(name) );
199
200         if( !detail::is_valid_parameter_name(name) ) {
201           BOOST_THROW_EXCEPTION( ConfigInvalidParameterNameError()
202                                  << errinfo_config_section(*current_section) << errinfo_config_parameter(name) );
203         }
204
205         std::map<std::string, std::set<std::string> >::iterator it;
206         it = parameters.find( *current_section );
207         assert( it != parameters.end() );
208
209         if( it->second.find(name) != it->second.end() ) {
210           BOOST_THROW_EXCEPTION( ConfigParameterRedefinedError()
211                                  << errinfo_config_section(*current_section) << errinfo_config_parameter(name) );
212         }
213
214         // Parameter "marker".
215         it->second.insert( name );
216
217         put( *current_section, name, value );
218       }
219       else
220         BOOST_THROW_EXCEPTION( ConfigLoneLineError() );
221     }
222   }
223   catch( boost::exception &e ) {
224     e << errinfo_config_line(lineno);
225     throw;
226   }
227 }
228
229
230 void
231 Configuration::import( const std::string &filename ) {
232   try {
233     std::ifstream file( filename.c_str() );
234     if( file.fail() )
235       BOOST_THROW_EXCEPTION( ConfigImportError() );
236     import( file );
237   }
238   catch( boost::exception &e ) {
239     e << errinfo_config_file(filename);
240     throw;
241   }
242 }
```

Listing B.10: src/error.hpp

```
1 #ifndef DICON_ERROR_HPP_
2 #define DICON_ERROR_HPP_
3
4 /*-- [Distribution] ------------------------------------------------
5  * This file is part of the Disease Control System DiCon.
```

```
 *
 * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
 * Designed and developed with the guidance of Nedialko B. Dimitrov
 * and Lauren Ancel Meyers at the University of Texas at Austin.
 *
 * DiCon is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * DiCon is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
 *----------------------------------------------------------------*/

/**
 * @file
 * @brief Exception related classes, macros and functions.
 */
#include <boost/exception.hpp>
#include <boost/version.hpp>


/**
 * @brief Define new @c errinfo structure.
 *
 * Define a new @c errinfo structure to be used with Boost's exception
 * mechanism (see Boost reference). The name of the new structure will
 * have the prefix @c errinfo_ and the suffix given by @e NAME (e.g.,
 * if @e NAME is @c value, the class will be named @c %errinfo_value).
 *
 * @param NAME Suffix of the new @c errinfo structure.
 * @param TYPE Type of the new @c errinfo structure.
 *
 * @see DICON_ERRINFO_TPL.
 */
#define DICON_ERRINFO(NAME, TYPE)                                       \
struct errinfo_##NAME                                                   \
  : boost::error_info<struct errinfo_##NAME##_, TYPE>                   \
{                                                                       \
  typedef boost::error_info<errinfo_##NAME##_, TYPE>                    \
    errinfo_t;                                                          \
  errinfo_##NAME( const TYPE &value )                                   \
    : boost::error_info<errinfo_##NAME##_, TYPE>(value) {}              \
}


/**
 * @brief Define new @c errinfo class with template.
 *
 * Define a new @c errinfo structure to be used with Boost's exception
 * mechanism (see Boost reference). In contrast to DICON_ERRINFO, the
 * structure defined by this macro will be templated with exactly one
 * parameter that gives the type of the value carried by the new @c
 * errinfo structure.
 *
 * The name of the new structure will have the prefix @c errinfo_ and
 * the suffix given by @e NAME (e.g., if @e NAME is @c value, the
 * class will be named @c %errinfo_value).
 *
 * @param NAME Suffix of the new @c errinfo structure.
 *
 * @see DICON_ERRINFO.
 */
#define DICON_ERRINFO_TPL(NAME)                                         \
template< typename T >                                                  \
struct errinfo_##NAME                                                   \
  : boost::error_info<struct errinfo_##NAME##_, T>                      \
{                                                                       \
  typedef boost::error_info<errinfo_##NAME##_, T>                       \
    errinfo_t;                                                          \
  errinfo_##NAME( const T &value )                                      \
    : boost::error_info<errinfo_##NAME##_, T>(value) {}                 \
}

#if BOOST_VERSION >= 104000
using boost::errinfo_errno;
using boost::errinfo_api_function;
#else
DICON_ERRINFO( errno, int );
DICON_ERRINFO( api_function, std::string );
#endif

/// Errinfo storing the pointer to another exception.
DICON_ERRINFO( nested_exception, boost::exception_ptr );

/// Templated errinfo storing the actual value.
DICON_ERRINFO_TPL( value );
/// Templated errinfo storing the minimum allowed value.
DICON_ERRINFO_TPL( min_value );
/// Templated errinfo storing the maximum allowed value.
DICON_ERRINFO_TPL( max_value );
/// Templated errinfo storing the expected value.
DICON_ERRINFO_TPL( expected_value );


/// Unspecified error.
struct Error : public virtual std::exception, public virtual boost::exception
{
  /// Get a general description of the error.
```

```
110     virtual const char *what() const throw() { return "Unspecified error.
        "; }
111 };
112
113
114 /// Assertion failed; invalid condition occurred.
115 struct AssertionError : virtual Error
116 { virtual const char *what() const throw() { return "Assertion failed; in
        valid condition occurred."; } };
117
118 /// System call or system-related call failed.
119 struct SystemError : virtual Error
120 { virtual const char *what() const throw() { return "System call or syst
        em-related call failed."; } };
121
122
123 /// Operation failed during I/O.
124 struct IOError : virtual SystemError
125 { virtual const char *what() const throw() { return "Operation failed dur
        ing I/O."; } };
126
127 /// Read failed while performing I/O.
128 struct IOReadError : virtual IOError
129 { virtual const char *what() const throw() { return "Read failed while pe
        rforming I/O."; } };
130
131 /// Write failed while performing I/O.
132 struct IOWriteError : virtual IOError
133 { virtual const char *what() const throw() { return "Write failed while p
        erforming I/O."; } };
134
135
136 /// Calculation did not succeed.
137 struct CalculationError : virtual Error
138 { virtual const char *what() const throw() { return "Calculation did not
        succeed."; } };
139
140 /// Calculation gave non-finite result.
141 struct CalculationNotFiniteError : virtual CalculationError
142 { virtual const char *what() const throw() { return "Calculation gave non
        -finite result."; } };
143
144
145 /// Unix signal related operation failed.
146 struct SignalError : virtual SystemError
147 { virtual const char *what() const throw() { return "Unix signal related
        operation failed."; } };
148
149 /// System time related operation failed.
150 struct TimeError : virtual SystemError
151 { virtual const char *what() const throw() { return "System time related
        operation failed."; } };
152
153
154 /**
155  * @brief Check if value is in range.
156  *
157  * Check if the given value is finite and within the given range. This
158  * function throws an exception of type AssertionError when @e value
159  * is less than @e minimum or greater than @e maximum.  The actual
160  * check performed is as follows.
161  *
162  * @code
163 if( !(minimum <= value && value <= maximum) ) {
164   // Throw AssertionError().
165 }
166 @endcode
167  *
168  * The resulting AssertionError will have the errinfo structures
169  * errinfo_value, errinfo_min_value, and errinfo_max_value set
170  * accordingly.
171  *
172  * @param value Value to check.
173  * @param minimum Minimum allowed value.
174  * @param maximum Maximum allowed value.
175  * @throws AssertionError when value is not within range.
176  */
177 template< typename T >
178 void DICON_ASSERT_RANGE( const T &value, const T &minimum, const T &
        maximum );
179
180
181 #include "error.ipp"
182
183 #endif //DICON_ERROR_HPP_
```

Listing B.11: src/error.ipp

```
1  // -*- c++ -*-
2
3  /*---- [Distribution] ------------------------------------------------
4   * This file is part of the Disease Control System DiCon.
5   *
6   * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
7   * Designed and developed with the guidance of Nedialko B. Dimitrov
8   * and Lauren Ancel Meyers at the University of Texas at Austin.
9   *
10  * DiCon is free software: you can redistribute it and/or modify it
11  * under the terms of the GNU General Public License as published by
12  * the Free Software Foundation, either version 3 of the License, or
13  * (at your option) any later version.
14  *
15  * DiCon is distributed in the hope that it will be useful, but
16  * WITHOUT ANY WARRANTY; without even the implied warranty of
17  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
18  * General Public License for more details.
19  *
20  * You should have received a copy of the GNU General Public License
21  * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
22  *--------------------------------------------------------------------*/
23
24 #include <cmath>
25
26
27 template< typename T >
28 inline
29 void
30 DICON_ASSERT_RANGE( const T &value, const T &minimum, const T &maximum
    ) {
```

```
31    if( !(std::isfinite(value) && minimum <= value && value <= maximum) ) {
32      BOOST_THROW_EXCEPTION( AssertionError()
33                             << errinfo_value<T>(value)
34                             << errinfo_min_value<T>(minimum)
35                             << errinfo_max_value<T>(maximum) );
36    }
37  }
```

Listing B.12: src/file.hpp

```
1   #ifndef DICON_FILE_HPP_
2   #define DICON_FILE_HPP_
3
4   /*---- [Distribution] ------------------------------------------------------
5    * This file is part of the Disease Control System DiCon.
6    *
7    * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8    * Designed and developed with the guidance of Nedialko B. Dimitrov
9    * and Lauren Ancel Meyers at the University of Texas at Austin.
10   *
11   * DiCon is free software: you can redistribute it and/or modify it
12   * under the terms of the GNU General Public License as published by
13   * the Free Software Foundation, either version 3 of the License, or
14   * (at your option) any later version.
15   *
16   * DiCon is distributed in the hope that it will be useful, but
17   * WITHOUT ANY WARRANTY; without even the implied warranty of
18   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19   * General Public License for more details.
20   *
21   * You should have received a copy of the GNU General Public License
22   * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23   *--------------------------------------------------------------------------*/
24
25  /**
26   * @file
27   * @brief File class.
28   */
29  #include "error.hpp"
30  #include <boost/filesystem/path.hpp>
31  #include <boost/noncopyable.hpp>
32  #include <boost/version.hpp>
33  #include <fcntl.h>
34  #include <string>
35
36
37  #if BOOST_VERSION >= 104000
38   using boost::errinfo_at_line;
39   using boost::errinfo_file_name;
40  #else
41  DICON_ERRINFO( at_line, int );
42  DICON_ERRINFO( file_name, std::string );
43  #endif
44
45  /// Errinfo storing the second filename.
46  DICON_ERRINFO( file_name_2, std::string );
47  /// Errinfo storing the file flags.
48  DICON_ERRINFO( file_flags , int          );
49  /// Errinfo storing the file mode.
50  DICON_ERRINFO( file_mode  , mode_t       );
51
52  /// Errinfo storing the file descriptor.
53  DICON_ERRINFO( file_descriptor  , int );
54  /// Errinfo storing the second file descriptor.
55  DICON_ERRINFO( file_descriptor_2, int );
56
57
58  /// Convert errinfo_file_flags to string.
59  std::string to_string( const errinfo_file_flags::errinfo_t &file_flags );
60  /// Convert errinfo_file_mode to string.
61  std::string to_string( const errinfo_file_mode::errinfo_t &file_mode );
62
63
64  /// %File or file system related error.
65  struct FileError : virtual IOError
66  { virtual const char *what() const throw() { return "File_or_file_system_related_error."; } };
67
68  /// Unix pipeline related error.
69  struct PipeError : virtual IOError
70  { virtual const char *what() const throw() { return "Unix_pipeline_related_error."; } };
71
72  /// Directory related error.
73  struct DirectoryError : virtual IOError
74  { virtual const char *what() const throw() { return "Directory_related_error."; } };
75
76  /// %Redirector related error.
77  struct RedirectorError : virtual IOError
78  { virtual const char *what() const throw() { return "Redirector_related_error."; } };
79
80
81  /// Read failed while operating on file.
82  struct FileReadError : public virtual IOReadError, public virtual FileError
83  { virtual const char *what() const throw() { return "Read_failed_while_operating_on_file."; } };
84
85  /// Write failed while operating on file.
86  struct FileWriteError : public virtual IOWriteError, public virtual FileError
87  { virtual const char *what() const throw() { return "Write_failed_while_operating_on_file."; } };
88
89
90  /// %File exists.
91  struct FileExistsError : virtual FileError
92  { virtual const char *what() const throw() { return "File_exists."; } };
93
94  /// Failed to open file.
95  struct FileOpenError : virtual FileError
96  { virtual const char *what() const throw() { return "Failed_to_open_file."; } };
97
```

```
98  /// Failed to create file.
99  struct FileCreateError : virtual FileError
100 { virtual const char *what() const throw() { return "Failed to create
    file."; } };
101
102 /// Failed to rename file.
103 struct FileRenameError : virtual FileError
104 { virtual const char *what() const throw() { return "Failed to rename
    file."; } };
105
106
107 /// %File not owned anymore.
108 struct FileNotOwnedError : virtual FileError
109 { virtual const char *what() const throw() { return "File not owned
    anymore."; } };
110
111
112 /**
113  * @brief %File access and file related operations.
114  *
115  * The File class provides access to a Unix file. The file can be
116  * opened for reading, writing, or read/write access, just like using
117  * the normal @c open system call. In fact, the @e flags and @e mode
118  * parameters in the constructor of File are exactly equivalent to the
119  * system call's respective parameters.
120  *
121  * Objects of the File class can be copied (by copy construction or by
122  * assignment). These copies share the same underlying file
123  * descriptor. When the last copy of a File object is destroyed, the
124  * underlying file descriptor is closed.
125  *
126  * In addition to opening named files, several static methods provide
127  * helper functionality, such as changing the current working
128  * directory, checking if a file exists, renaming files, getting
129  * unique or temporary filenames, etc.
130  */
131 class File {
132 public:
133   /// Default file flags used in the constructor.
134   static const int default_flags = O_RDONLY;
135   /// Default file mode used in the constructor.
136   static const mode_t default_mode = (S_IRUSR | S_IWUSR) | (S_IRGRP |
    S_IWGRP) | (S_IROTH | S_IWOTH);
137
138   /**
139    * @brief Open named file.
140    *
141    * Constructor that opens the named file given by @e filename for
142    * reading, writing, or read/write access. The exact mode depends on
143    * the parameter @e flags. In case the file is to be created (@e
144    * flags contains @c O_CREAT), the parameter @e mode describes the
145    * permissions to use in case the file has to be created.
146    *
147    * If the file is to be created (as indicated by @c O_CREAT), any
148    * directories leading to the file specified by @e filename are
149    * automatically created as well.
150    *
151    * @param filename Filename.
152    * @param flags %File flags (see @c open system call).
153    * @param mode %File mode (see @c open system call).
154    * @throws FileOpenError when the file could not be opened (and @e
155    *   flags does not contain @c O_CREAT).
156    * @throws FileCreateError when the file could not be created (and
157    *   @e flags contains @c O_CREAT but not @c O_EXCL).
158    * @throws FileExistsError when the file already exists (and @e
159    *   flags contains both @c O_CREAT and @c O_EXCL flags).
160    */
161   File( const boost::filesystem::path &filename, int flags = O_RDONLY,
    mode_t mode = 0 );
162
163 public:
164   /**
165    * @brief Change working directory.
166    *
167    * Change the current working directory to the directory given by @e
168    * dirname.
169    *
170    * @param dirname Directory name.
171    * @throws DirectoryError when the directory could not be changed.
172    */
173   static void chdir( const boost::filesystem::path &dirname );
174
175 public:
176   /**
177    * @brief Check if file exists.
178    *
179    * Check if the file or directory given by @e path exists. This call
180    * is guaranteed to not throw an exception.
181    *
182    * @param path %File or directory name.
183    * @returns true iff the file or directory exists.
184    */
185   static bool exists( const boost::filesystem::path &path ) throw();
186   /**
187    * @brief Rename file.
188    *
189    * Rename the file or directory given by @e old_path to the new name
190    * given by @e new_path. If neither @e old_path nor @e new_path are
191    * directories and the target already exists, it will be overwritten
192    * without throwing an exception. This can be used to atomically
193    * replace files in place.
194    *
195    * @param old_path Old file or directory name.
196    * @param new_path New file or directory name.
197    * @throws FileRenameError when the file or directory could not be
198    *   renamed.
199    */
200   static void rename( const boost::filesystem::path &old_path, const
    boost::filesystem::path &new_path );
201
202 public:
203   /// Default file flags used in unique(), unique_name(),
    unique_temporary().
204   static const int default_flags_unique = O_WRONLY | O_APPEND |
    O_CREAT | O_EXCL;
205   /// Default file mode used in unique(), unique_name(),
    unique_temporary().
206   static const mode_t default_mode_unique = default_mode;
207
208   /**
209    * @brief Create file with unique name.
210    *
```

```
211     * Create a new File object representing a file with a unique name
212     * that is not yet in use. If the file given by @e filename does not
213     * exist, it will be used without modification. Otherwise,
214     * increasing suffixes of the form <code>.[number]</code> will be
215     * added to the filename until an available filename is found (e.g.,
216     * if @e filename is <code>file.txt</code> and a file with this name
217     * already exists, the function will try the filenames
218     * <code>file.txt.1</code>, <code>file.txt.2</code>, etc. until a
219     * free filename is found).
220     *
221     * As with File(), this function creates all missing directories in
222     * @e filename when a new file is created.
223     *
224     * @note @e flags should contain both @c O_CREAT and @c
225     * O_EXCL. Otherwise, this function will not be able to check if a
226     * file already exists and might overwrite an existing file or throw
227     * an unexpected exception.
228     *
229     * @param filename Filename, or filename prefix in case file already
230     *   exists.
231     * @param flags %File flags (see @c open system call), with @c
232     *   O_CREAT, @c O_EXCL.
233     * @param mode %File mode (see @c open system call).
234     * @returns File object representing the newly created file.
235     * @throws FileOpenError when the file could not be opened (and @e
236     *   flags does not contain @c O_CREAT).
237     * @throws FileCreateError when the file could not be created (and
238     *   @e flags contains @c O_CREAT but not @c O_EXCL).
239     */
240    static File unique( const boost::filesystem::path &filename,
241                        int flags = default_flags_unique, mode_t mode =
    default_mode_unique );
242
243    /**
244     * @brief Create unique file and get name.
245     *
246     * Create a unique file and return its name. This is similar to
247     * unique() but directly returns the filename instead of a File
248     * object to the newly created, unique file. This call still creates
249     * the file so that future calls will not return the same filename.
250     *
251     * @see unique().
252     *
253     * @param filename Filename, or filename prefix in case file already
254     *   exists.
255     * @param flags %File flags (see @c open system call), with @c
256     *   O_CREAT, @c O_EXCL.
257     * @param mode %File mode (see @c open system call).
258     * @returns File object representing the newly created file.
259     * @throws FileOpenError when the file could not be opened (and @e
260     *   flags does not contain @c O_CREAT).
261     * @throws FileCreateError when the file could not be created (and
262     *   @e flags contains @c O_CREAT but not @c O_EXCL).
263     */
264    static boost::filesystem::path unique_name( const boost::filesystem
    ::path &filename,
265                                                int flags =
    default_flags_unique, mode_t mode = default_mode_unique );
266
267    /**
268     * @brief Create unique temporary file and get name.
269     *
270     * Create a unique file and return its name. The name of the newly
271     * created file is derived from @e filename by applying a template
272     * of the form <code>.[filename]_tmp</code> to the filename part of
273     * @e filename, i.e., any leading directories are kept the same.
274     * Apart from the filename mangling, this function behaves exactly
275     * as unique_name(), i.e., as long as @e flags contains both @c
276     * O_CREAT and @c O_EXCL, a unique filename is returned, applying a
277     * suffix of the form <code>.[number]</code> to the mangled
278     * filename.
279     *
280     * As the name suggests this function can be used to create unique
281     * temporary file names, e.g., when first writing data to a file,
282     * and then renaming this temporary file to the final name when all
283     * data has been written.
284     *
285     * @see unique_name().
286     *
287     * @param filename Base filename used to derive the temporary
288     *   filename.
289     * @param flags %File flags (see @c open system call), with @c
290     *   O_CREAT, @c O_EXCL.
291     * @param mode %File mode (see @c open system call).
292     * @returns File object representing the newly created file.
293     * @throws FileOpenError when the file could not be opened (and @e
294     *   flags does not contain @c O_CREAT).
295     * @throws FileCreateError when the file could not be created (and
296     *   @e flags contains @c O_CREAT but not @c O_EXCL).
297     */
298    static boost::filesystem::path unique_temporary( const boost::
    filesystem::path &filename,
299                                                     int flags =
    default_flags_unique, mode_t mode = default_mode_unique );
300
301  public:
302    /**
303     * @brief Get file descriptor.
304     *
305     * Get the underlying Unix file descriptor to the file represented
306     * by this File object. This call does not change ownership to the
307     * file descriptor.
308     *
309     * @see file(bool).
310     *
311     * @returns File descriptor.
312     * @throws FileNotOwnedError when this File object does not own a
313     *   file anymore.
314     */
315    int file() const;
316    /**
317     * @brief Get file descriptor, possibly taking ownership.
318     *
319     * Get the underlying Unix file descriptor to the file represented
320     * by this File object. If @e take_ownership is @c true, then the
321     * object (and all copies of the object) will lose ownership of the
322     * file, i.e., the underlying file will not be closed when the last
323     * copy of the object is destroyed; also, future calls to file() or
324     * file(bool) will throw an exception.
325     *
326     * @param take_ownership Set to @c true when caller wants to take
327     *   ownership of file.
```

```
328     * @returns File descriptor.
329     * @throws FileNotOwnedError when this File object does not own a
330     *   file anymore.
331     */
332    int file( bool take_ownership = false );
333
334    /**
335     * @brief Get filename.
336     *
337     * Get the filename of this object. This is equivalent to the
338     * filename originally specified when creating the object using the
339     * constructor, or the filename finally settled on when using
340     * unique().
341     *
342     * @returns Filename.
343     */
344    boost::filesystem::path filename() const;
345
346  private:
347    struct Data
348      : boost::noncopyable
349    {
350      Data( const boost::filesystem::path &filename );
351      ~Data();
352
353      int filefd;
354      bool owned;
355      boost::filesystem::path filename;
356    };
357
358    boost::shared_ptr<Data> data_;
359  };
360
361  #endif //DICON_FILE_HPP_
```

Listing B.13: src/file.cpp

```
1   /*--- [Distribution] ---------------------------------------------------
2    * This file is part of the Disease Control System DiCon.
3    *
4    * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
5    * Designed and developed with the guidance of Nedialko B. Dimitrov
6    * and Lauren Ancel Meyers at the University of Texas at Austin.
7    *
8    * DiCon is free software: you can redistribute it and/or modify it
9    * under the terms of the GNU General Public License as published by
10   * the Free Software Foundation, either version 3 of the License, or
11   * (at your option) any later version.
12   *
13   * DiCon is distributed in the hope that it will be useful, but
14   * WITHOUT ANY WARRANTY; without even the implied warranty of
15   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
16   * General Public License for more details.
17   *
18   * You should have received a copy of the GNU General Public License
19   * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
20   *-----------------------------------------------------------------*/
21
22  #include "file.hpp"
23  #include <boost/filesystem/convenience.hpp>
24  #include <boost/format.hpp>
25  #include <cerrno>
26  #include <cstdio>
27  #include <sstream>
28
29
30  std::string
31  to_string( const errinfo_file_flags::errinfo_t &file_flags ) {
32    int flags = file_flags.value();
33    std::stringstream str;
34
35    str << flags
36        << ", ";
37
38    std::ios_base::fmtflags ioflags = str.flags( std::ios_base::oct );
39    str << '0'
40        << flags;
41    str.flags( ioflags );
42    str << ", ";
43
44    bool first = true;
45
46  #define DICON_CHECK_FILE_FLAGS_(X, FORCE)                    \
47    if( (X != 0 || FORCE) && (flags & X) == X ) do {           \
48      if( !first ) str << '|'; else first = false;             \
49      str << #X; flags ^= X;                                   \
50      } while( false )                                         \
51        /**/
52
53    /**/ DICON_CHECK_FILE_FLAGS_( O_RDWR  , true );
54    else DICON_CHECK_FILE_FLAGS_( O_WRONLY, true );
55    else DICON_CHECK_FILE_FLAGS_( O_RDONLY, true );
56
57    DICON_CHECK_FILE_FLAGS_( O_CREAT , false );
58    DICON_CHECK_FILE_FLAGS_( O_EXCL  , false );
59    DICON_CHECK_FILE_FLAGS_( O_NOCTTY, false );
60    DICON_CHECK_FILE_FLAGS_( O_TRUNC , false );
61
62    DICON_CHECK_FILE_FLAGS_( O_APPEND    , false );
63    DICON_CHECK_FILE_FLAGS_( O_ASYNC     , false );
64    DICON_CHECK_FILE_FLAGS_( O_DIRECTORY, false );
65    DICON_CHECK_FILE_FLAGS_( O_LARGEFILE, false );
66    DICON_CHECK_FILE_FLAGS_( O_NOFOLLOW , false );
67    DICON_CHECK_FILE_FLAGS_( O_NONBLOCK , false );
68    DICON_CHECK_FILE_FLAGS_( O_NDELAY   , false );
69    DICON_CHECK_FILE_FLAGS_( O_SYNC     , false );
70
71  #undef DICON_CHECK_FILE_FLAGS_
72
73    if( flags != 0 ) {
74      if( !first )
75        str << '|';
76
77      std::ios_base::fmtflags ioflags = str.flags( std::ios_base::oct );
78      str << '0'
79          << flags;
80      str.flags( ioflags );
81    }
82
83    return str.str();
```

```
84  }
85
86
87  std::string
88  to_string( const errinfo_file_mode::errinfo_t &file_mode ) {
89    std::stringstream str;
90
91    str << file_mode.value()
92        << ",";
93
94    std::ios_base::fmtflags ioflags = str.flags( std::ios_base::oct );
95    str << '0'
96        << file_mode.value();
97    str.flags( ioflags );
98
99    return str.str();
100 }
101
102
103 File::Data::Data( const boost::filesystem::path &filename )
104   : filefd(-1), owned(true), filename(filename)
105 {
106 }
107
108
109 File::Data::~Data() {
110   if( owned && filefd != -1 ) {
111     close( filefd );
112     filefd = -1;
113   }
114 }
115
116
117 File::File( const boost::filesystem::path &filename, int flags, mode_t
    mode )
118   : data_(new Data(filename))
119 {
120   std::string file_string = filename.file_string();
121
122   if( (flags & O_CREAT) == 0 ) {
123     // Open file (without mode parameter).
124
125     if( (data_->filefd = ::open(file_string.c_str(), flags)) == -1 ) {
126       BOOST_THROW_EXCEPTION( FileOpenError() << errinfo_api_function("
    open")
127                              << errinfo_errno(errno) <<
    errinfo_file_name(file_string) << errinfo_file_flags(flags) );
128     }
129   }
130   else {
131     // Optionally create file (with mode parameter).
132
133     boost::filesystem::create_directories( filename.parent_path() );
134
135     if( (data_->filefd = ::open(file_string.c_str(), flags, mode)) ==
    -1 ) {
136       if( (flags & O_EXCL) != 0 && errno == EEXIST ) {
137         BOOST_THROW_EXCEPTION( FileExistsError() <<
    errinfo_api_function("open")
138                                << errinfo_errno(errno) <<
    errinfo_file_name(file_string)
139                                << errinfo_file_flags(flags) <<
    errinfo_file_mode(mode) );
140       }
141       else {
142         BOOST_THROW_EXCEPTION( FileCreateError() <<
    errinfo_api_function("open")
143                                << errinfo_errno(errno) <<
    errinfo_file_name(file_string)
144                                << errinfo_file_flags(flags) <<
    errinfo_file_mode(mode) );
145       }
146     }
147   }
148 }
149
150
151 void
152 File::chdir( const boost::filesystem::path &dirname ) {
153   const std::string &directory = dirname.file_string();
154
155   if( ::chdir(directory.c_str()) != 0 ) {
156     BOOST_THROW_EXCEPTION( DirectoryError() << errinfo_errno(errno)
157                            << errinfo_api_function("chdir") <<
    errinfo_file_name(directory) );
158   }
159 }
160
161
162 bool
163 File::exists( const boost::filesystem::path &path ) throw() {
164   // The Boost implementation bfs::exists throws exceptions, e.g.,
165   // when the filename is invalid (disk drive or directory doesn't
166   // exist etc.). But we don't want any exceptions to be thrown.
167
168   struct stat buffer;
169
170   return stat(path.file_string().c_str(), &buffer) == 0;
171 }
172
173
174 void
175 File::rename( const boost::filesystem::path &old_path, const boost::
    filesystem::path &new_path ) {
176 #if 0
177   // In contrast to Boost documentation, this (as of version 1.40.0)
178   // throws an exception in case 'new_path' exists. For this reason,
179   // using bfs::rename is not possible at the moment (exception must
180   // not be thrown, but 'new_path' must be overwritten in-place).
181
182   boost::filesystem::rename( tmp_file, file );
183 #else
184   if( ::rename(old_path.file_string().c_str(), new_path.file_string().
    c_str()) != 0 ) {
185     BOOST_THROW_EXCEPTION( FileRenameError() << errinfo_api_function("
    rename") << errinfo_errno(errno)
186                            << errinfo_file_name(old_path.file_string()
    ) << errinfo_file_name_2(new_path.file_string()) );
187   }
188 #endif
189 }
190
```

```
191
192 File
193 File::unique( const boost::filesystem::path &filename, int flags,
    mode_t mode ) {
194   for( unsigned i = 0;; ++i ) {
195     boost::filesystem::path name
196       = i ? filename.parent_path() / (boost::format("%s.%u") %
    filename.filename() % i).str() : filename;
197
198     try {
199       return File( name, flags, mode );
200     }
201     catch( FileExistsError & ) {
202       // Try next name.
203       continue;
204     }
205   }
206 }
207
208
209 boost::filesystem::path
210 File::unique_name( const boost::filesystem::path &filename, int flags,
    mode_t mode ) {
211   return unique( filename, flags, mode ).filename();
212 }
213
214
215 boost::filesystem::path
216 File::unique_temporary( const boost::filesystem::path &filename, int
    flags, mode_t mode ) {
217   return unique_name( filename.parent_path() / (boost::format(".%s_tmp
    ") % filename.filename()).str(), flags, mode );
218 }
219
220
221 int
222 File::file() const {
223   if( !data_->owned ) {
224     BOOST_THROW_EXCEPTION( FileNotOwnedError()
225                            << errinfo_file_descriptor(data_->filefd) )
    ;
226   }
227
228   return data_->filefd;
229 }
230
231
232 int
233 File::file( bool take_ownership ) {
234   if( !data_->owned ) {
235     BOOST_THROW_EXCEPTION( FileNotOwnedError()
236                            << errinfo_file_descriptor(data_->filefd) )
    ;
237   }
238
239   if( take_ownership )
240     data_->owned = false;
241
242   return data_->filefd;
243 }
244
```

```
245
246 boost::filesystem::path
247 File::filename() const {
248   return data_->filename;
249 }
```

Listing B.14: src/job.hpp

```
1  #ifndef DICON_JOB_HPP_
2  #define DICON_JOB_HPP_
3
4  /*--- [Distribution] ------------------------------------------------
5   * This file is part of the Disease Control System DiCon.
6   *
7   * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8   * Designed and developed with the guidance of Nedialko B. Dimitrov
9   * and Lauren Ancel Meyers at the University of Texas at Austin.
10  *
11  * DiCon is free software: you can redistribute it and/or modify it
12  * under the terms of the GNU General Public License as published by
13  * the Free Software Foundation, either version 3 of the License, or
14  * (at your option) any later version.
15  *
16  * DiCon is distributed in the hope that it will be useful, but
17  * WITHOUT ANY WARRANTY; without even the implied warranty of
18  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19  * General Public License for more details.
20  *
21  * You should have received a copy of the GNU General Public License
22  * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23  *-------------------------------------------------------------------*/
24
25 /**
26  * @file
27  * @brief Job structure.
28  */
29 #include "lazy/set.hpp"
30 #include "message.hpp"
31 #include <boost/filesystem/path.hpp>
32 #include <boost/shared_ptr.hpp>
33
34
35 /**
36  * @brief %Job descriptor.
37  *
38  * The Job structure holds the main information associated with each
39  * job.  This information is normally not copied but rather passed
40  * around by the means of a smart pointer, as referenced by
41  * Job::ptr_t.
42  */
43 struct Job {
44   /// Pointer to Job structure.
45   typedef boost::shared_ptr<Job> ptr_t;
46
47   /// Unique ID of job (>= 1).
48   job_id_t id;
49
50   /// Number of job set job belongs to (>= 1).
51   size_t job_set_id;
```

```
52    /// Number of argument set within job set job belongs to (>= 1).
53    size_t arg_set_id;
54
55    /// Maximum allowed number of simulator nodes (0 = unlimited).
56    unsigned max_nodes;
57    /// Maximum allowed number of simulations (0 = unlimited).
58    simcount_t max_sims;
59    /// Size of job backlog queue.
60    unsigned job_backlog;
61    /// Size of node backlog queue, per node.
62    unsigned node_backlog;
63
64    /// Maximum number of seconds between checkpoints (0 = disabled).
65    unsigned checkpoint_secs;
66    /// Maximum number of simulations between checkpoints (0 = disabled)
      .
67    unsigned checkpoint_sims;
68    /// Pointer to lazy set containing absolute simulation counts for
      checkpoint.
69    boost::shared_ptr<LazySet<simcount_t> > checkpoints;
70
71    /// Path to job directory.
72    boost::filesystem::path job_directory;
73
74    /// Path to job logfile within job directory.
75    boost::filesystem::path job_logfile;
76    /// Log level to use for job logfile.
77    LogLevel job_log_level;
78    /// Path to checkpoint file within job directory.
79    boost::filesystem::path checkpoint_file;
80    /// @c printf mask (with <code>\%u</code>: node number) for
      optimizer logfile within job directory.
81    std::string opt_logfile_mask;
82    /// @c printf mask (with <code>\%u</code>: node number) for
      simulator logfile within job directory.
83    std::string sim_logfile_mask;
84
85    /// @c printf mask (with <code>\%u</code>: simulation count) for
      optimizer map dumpfile within job directory.
86    std::string optimizer_map_file_mask;
87    /// @c printf mask (with <code>\%u</code>: simulation count) for
      optimizer lib dumpfile within job directory.
88    std::string optimizer_lib_file_mask;
89    /// @c printf mask (with <code>\%u</code>: simulation count) for
      binary policy file within job directory.
90    std::string policy_bin_dumpfile_mask;
91    /// @c printf mask (with <code>\%u</code>: simulation count) for
      textual policy file within job directory.
92    std::string policy_txt_dumpfile_mask;
93
94    /// Number of policies to output in policy files.
95    unsigned policy_count;
96    /// Path to binary policy result file within job directory.
97    boost::filesystem::path policy_bin_result;
98    /// Path to textual policy result file within job directory.
99    boost::filesystem::path policy_txt_result;
100
101   /// Path to optimizer library.
102   std::string optimizer_library;
103   /// Arguments to optimizer library.
104   arguments_t optimizer_arguments;
105
106   /// Path to simulator command.
107   std::string simulator_command;
108   /// Arguments to simulator command.
109   arguments_t simulator_arguments;
110 };
111
112 #endif //DICON_JOB_HPP_
```

Listing B.15: src/job_map.hpp

```
1   #ifndef DICON_JOB_MAP_HPP_
2   #define DICON_JOB_MAP_HPP_
3
4   /*---- [Distribution] ------------------------------------------------
5    * This file is part of the Disease Control System DiCon.
6    *
7    * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8    * Designed and developed with the guidance of Nedialko B. Dimitrov
9    * and Lauren Ancel Meyers at the University of Texas at Austin.
10   *
11   * DiCon is free software: you can redistribute it and/or modify it
12   * under the terms of the GNU General Public License as published by
13   * the Free Software Foundation, either version 3 of the License, or
14   * (at your option) any later version.
15   *
16   * DiCon is distributed in the hope that it will be useful, but
17   * WITHOUT ANY WARRANTY; without even the implied warranty of
18   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19   * General Public License for more details.
20   *
21   * You should have received a copy of the GNU General Public License
22   * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23   *--------------------------------------------------------------------*/
24
25  /**
26   * @file
27   * @brief JobMap class.
28   */
29  #include "job.hpp"
30  #include <boost/noncopyable.hpp>
31  #include <deque>
32  #include <map>
33  #include <set>
34
35
36  /**
37   * @brief Mapping of jobs to processing nodes.
38   *
39   * The templated JobMap class provides the mapping between jobs (Job
40   * structure) and processing nodes (@c int representing the node in
41   * the corresponding MPI communicator context), in the main node
42   * manager.  It allows to associate processing nodes with jobs, either
43   * as %optimizer or simulator nodes, and provides user-defined
44   * additional structures per job (and thus per %optimizer node) and
45   * per simulator node associated with a job. The types of these
46   * user-defined structures are given by @c JobInfoT and @c NodeInfoT,
47   * respectively.
48   *
```

```
 * Each job, when it is currently active in the system, is associated
 * with exactly one %optimizer node, and an arbitrary number of
 * simulator nodes (only limited by the number of nodes in the MPI
 * communicator), though typically each job has at least one simulator
 * node (the only exceptions being jobs with combined
 * %optimizer/simulator nodes, and jobs that have already finished and
 * are only dumping results).
 *
 * The JobMap class provides methods to look up the job a node is
 * currently associated with, and get appropriate job and node
 * information for a job or node. It also keeps track of nodes that
 * have been registered but are not associated with any job.
 *
 * Any node in the job map must be associated with at most one job at
 * any time (i.e., nodes may not share a job). As jobs (Job structure)
 * are typically shared by the means of smart pointers, equality of
 * jobs is compared in terms of pointer equality, i.e., only identical
 * job objects are considered equal. In particular, the job ID is not
 * used when comparing jobs.
 *
 * @tparam JobInfoT Type of additional structure per active job (and
 *   thus per %optimizer node).
 * @tparam NodeInfoT Type of additional structure per each simulator
 *   node associated with job.
 *
 * @see MainNodeManager, MainNodeManagerImpl.
 */
template< typename JobInfoT, typename NodeInfoT >
class JobMap
  : boost::noncopyable
{
public:
  /**
   * @brief Register processing node.
   *
   * Register the processing node given by @e node with the job
   * map. This method must be called before any of the other methods
   * accept the given node as a parameter.
   *
   * @param node Node in the MPI communicator context.
   * @throws AssertionError when node is already registered.
   */
  void register_node( int node );

  /**
   * @brief Create new job.
   *
   * Create new job given by @e job with %optimizer node given by @e
   * optimizer_node. The job must not be active, and the %optimizer
   * node must not be registered with any other job, when calling this
   * method.
   *
   * A new object of the user-defined job structure given by template
   * argument @e JobInfoT is default-constructed when calling this
   * method.
   *
   * @param job %Job.
   * @param optimizer_node %Optimizer node.
   * @throws AssertionError when %optimizer node is not registered or
   *   already assigned to another job, or job is already active.
   */
  void create_job( const Job::ptr_t &job, int optimizer_node );
  /**
   * @brief Assign simulator node to job.
   *
   * Assign simulator node given by @e simulator_node to job given by
   * @e job. The job must be active, and the simulator node must not
   * be associated with any other job, when calling this method.
   *
   * A new object of the user-defined node structure given by template
   * argument @e NodeInfoT is default-constructed when calling this
   * method.
   *
   * @param job %Job.
   * @param simulator_node %Simulator node.
   * @throws AssertionError when simulator node is not registered or
   *   already assigned to another job, or job is not active.
   */
  void assign_node( const Job::ptr_t &job, int simulator_node );

  /**
   * @brief Finish active job.
   *
   * Finish the job given by @e job and @e optimizer_node. This
   * removes the job from the job map, so that it is not considered
   * active anymore. Any simulator nodes associated with the job must
   * be removed with release_node() prior to calling this method. The
   * %optimizer node will be returned to the pool of unassociated
   * nodes.
   *
   * @param job %Job.
   * @param optimizer_node %Optimizer node.
   * @throws AssertionError when %optimizer node is not registered or
   *   is not the %optimizer node associated with the given job, or
   *   the job is not active or has any simulator nodes still
   *   assigned.
   */
  void finish_job( const Job::ptr_t &job, int optimizer_node );
  /**
   * @brief Release simulator node from job.
   *
   * Release the simulator node given by @e simulator_node from the
   * job given by @e job. This returns the given simulator node to the
   * pool of unassociated nodes.
   *
   * @param job %Job.
   * @param simulator_node %Simulator node.
   * @throws AssertionError when simulator node is not registered or
   *   is not an associated simulator node with the given job, or the
   *   job is not active.
   */
  void release_node( const Job::ptr_t &job, int simulator_node );

public:
  /**
   * @brief Get free node count.
   *
   * Get the number of currently unassociated nodes in the job map.
   * This is equivalent to the number of nodes previously registered
   * with register_node(), minus the number of registered nodes
   * currently associated with a job (either as %optimizer or
   * simulator nodes).
```

```
171     *
172     * @returns Number of unassociated nodes.
173     */
174    unsigned free_node_count() const;
175    /**
176     * @brief Get unassociated node.
177     *
178     * Get an unassociated node from the job map. This returns the node
179     * number (@c int representing the node in the corresponding MPI
180     * communicator context) of an unassociated node in the job map. If
181     * no such node is available, this call fails.
182     *
183     * @note This method does not remove the returned node from the pool
184     * of unassociated nodes. Therefore, two consecutive invocations of
185     * this method are allowed to return the same node.
186     *
187     * @returns Unassociated node.
188     * @throws AssertionError when no unassociated node is available.
189     *
190     * @see free_node_count().
191     */
192    int get_free_node() const;
193
194    /**
195     * @brief Get active jobs.
196     *
197     * Get the list of jobs currently active in the system. This is the
198     * list of all jobs previously created with create_job() and not yet
199     * removed with finish_job().
200     *
201     * @returns List of active jobs.
202     */
203    std::vector<Job::ptr_t> jobs() const;
204
205    /**
206     * @brief Get job associated with node.
207     *
208     * Get the job currently associated with the node given by @e
209     * node. If this node does not have a job associated, this call
210     * fails.
211     *
212     * @param node Node.
213     * @returns %Job associated with node.
214     * @throws AssertionError when the node is not registered or has no
215     *   job associated.
216     */
217    Job::ptr_t job( int node ) const;
218
219  public:
220    /**
221     * @brief Get %optimizer node of job.
222     *
223     * Get the %optimizer node of the job given by @e job. If this job
224     * is not active, this call fails.
225     *
226     * @param job %Job.
227     * @returns %Optimizer node associated with job.
228     * @throws AssertionError when job is not active.
229     */
230    int optimizer_node( const Job::ptr_t &job ) const;
231    /**
232     * @brief Get simulator nodes of job.
233     *
234     * Get the list of simulator nodes of the job given by @e job. If
235     * this job is not active, this call fails.
236     *
237     * @param job %Job.
238     * @returns List of simulator nodes associated with job.
239     * @throws AssertionError when job is not active.
240     */
241    std::vector<int> simulator_nodes( const Job::ptr_t &job ) const;
242
243  public:
244    /// Get user-defined job structure.
245    const JobInfoT &job_info( const Job::ptr_t &job ) const;
246    /**
247     * @brief Get user-defined job structure.
248     *
249     * Get the user-defined structure for the job given by @e job. If
250     * this job is not active, this call fails.
251     *
252     * @param job %Job.
253     * @returns User-defined job structure.
254     * @throws AssertionError when job is not active.
255     */
256    JobInfoT &job_info( const Job::ptr_t &job );
257
258    /// Get user-defined node structure.
259    const NodeInfoT &node_info( const Job::ptr_t &job, int
   simulator_node ) const;
260    /**
261     * @brief Get user-defined node structure.
262     *
263     * Get the user-defined structure for the simulator node given by @e
264     * simulator_node associated with the job given by @e job. If this
265     * job is not active, or the given node not a simulator node
266     * associated with the job, this call fails.
267     *
268     * @param job %Job.
269     * @param simulator_node %Simulator node.
270     * @returns User-defined node structure.
271     * @throws AssertionError when simulator node is not registered, or
272     *   is not a simulator node associated with the given job, or job
273     *   is not active.
274     */
275    NodeInfoT &node_info( const Job::ptr_t &job, int simulator_node );
276
277  private:
278    struct NodeInfo {
279      NodeInfoT node_info;
280    };
281
282    typedef std::map<int, NodeInfo> simulator_nodes_t;
283
284    struct JobInfo {
285      simulator_nodes_t simulator_nodes;
286      int optimizer_node;
287      JobInfoT job_info;
288    };
289
290    typedef std::map<int, Job::ptr_t> nodes_t;
291    nodes_t nodes_;
```

```
292
293   typedef std::map<Job::ptr_t, JobInfo> jobs_t;
294   jobs_t jobs_;
295 };
296
297
298 #include "job_map.ipp"
299
300 #endif //DICON_JOB_MAP_HPP_
```

Listing B.16: src/job_map.ipp

```
1  // -*- c++ -*-
2
3  /*---- [Distribution] ------------------------------------------------
4   * This file is part of the Disease Control System DiCon.
5   *
6   * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
7   * Designed and developed with the guidance of Nedialko B. Dimitrov
8   * and Lauren Ancel Meyers at the University of Texas at Austin.
9   *
10  * DiCon is free software: you can redistribute it and/or modify it
11  * under the terms of the GNU General Public License as published by
12  * the Free Software Foundation, either version 3 of the License, or
13  * (at your option) any later version.
14  *
15  * DiCon is distributed in the hope that it will be useful, but
16  * WITHOUT ANY WARRANTY; without even the implied warranty of
17  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
18  * General Public License for more details.
19  *
20  * You should have received a copy of the GNU General Public License
21  * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
22  *--------------------------------------------------------------------*/
23
24 #include "error.hpp"
25 #include <boost/foreach.hpp>
26
27
28 template< typename JobInfoT, typename NodeInfoT >
29 inline
30 void
31 JobMap<JobInfoT, NodeInfoT>::register_node( int node ) {
32   if( !nodes_.insert(std::make_pair(node, Job::ptr_t())).second )
33     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<int>(node
  ) );
34 }
35
36
37 template< typename JobInfoT, typename NodeInfoT >
38 inline
39 void
40 JobMap<JobInfoT, NodeInfoT>::create_job( const Job::ptr_t &job, int
  optimizer_node ) {
41   typename nodes_t::iterator node_it;
42   if( (node_it = nodes_.find(optimizer_node)) == nodes_.end() )
43     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<int>(
  optimizer_node) );
44   if( jobs_.find(job) != jobs_.end() )
45     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<job_id_t
  >(job->id) );
46
47   if( node_it->second )
48     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<int>(
  optimizer_node) );
49
50   node_it->second = job;
51
52   // Create new job record.
53   JobInfo &info = jobs_[job];
54   info.optimizer_node = optimizer_node;
55 }
56
57
58 template< typename JobInfoT, typename NodeInfoT >
59 inline
60 void
61 JobMap<JobInfoT, NodeInfoT>::assign_node( const Job::ptr_t &job, int
  simulator_node ) {
62   typename jobs_t::iterator job_it;
63   if( (job_it = jobs_.find(job)) == jobs_.end() )
64     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<job_id_t
  >(job->id) );
65   typename nodes_t::iterator node_it;
66   if( (node_it = nodes_.find(simulator_node)) == nodes_.end() )
67     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<int>(
  simulator_node) );
68
69   if( node_it->second )
70     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<int>(
  simulator_node) );
71
72   node_it->second = job;
73
74   // Add node to job record.
75   job_it->second.simulator_nodes[simulator_node];
76 }
77
78
79 template< typename JobInfoT, typename NodeInfoT >
80 inline
81 void
82 JobMap<JobInfoT, NodeInfoT>::finish_job( const Job::ptr_t &job, int
  optimizer_node ) {
83   typename jobs_t::iterator job_it;
84   if( (job_it = jobs_.find(job)) == jobs_.end() )
85     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<job_id_t
  >(job->id) );
86   typename nodes_t::iterator node_it;
87   if( (node_it = nodes_.find(optimizer_node)) == nodes_.end() )
88     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<int>(
  optimizer_node) );
89
90   if( job_it->second.optimizer_node != optimizer_node )
91     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<int>(
  optimizer_node) );
92   if( !job_it->second.simulator_nodes.empty() )
93     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<job_id_t
  >(job->id) );
94
```

```
95    jobs_.erase( job_it );
96    node_it->second.reset();
97  }
98
99
100 template< typename JobInfoT, typename NodeInfoT >
101 inline
102 void
103 JobMap<JobInfoT, NodeInfoT>::release_node( const Job::ptr_t &job, int
    simulator_node ) {
104   typename jobs_t::iterator job_it;
105   if( (job_it = jobs_.find(job)) == jobs_.end() )
106     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<job_id_t
    >(job->id) );
107   typename nodes_t::iterator node_it;
108   if( (node_it = nodes_.find(simulator_node)) == nodes_.end() )
109     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<int>(
    simulator_node) );
110   typename simulator_nodes_t::iterator sim_node_it;
111   if( (sim_node_it = job_it->second.simulator_nodes.find(
    simulator_node)) == job_it->second.simulator_nodes.end() )
112     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<int>(
    simulator_node) );
113
114   job_it->second.simulator_nodes.erase( sim_node_it );
115   node_it->second.reset();
116 }
117
118
119 template< typename JobInfoT, typename NodeInfoT >
120 inline
121 unsigned
122 JobMap<JobInfoT, NodeInfoT>::free_node_count() const {
123   unsigned count = 0;
124
125   BOOST_FOREACH( const nodes_t::value_type &entry, nodes_ ) {
126     if( !entry.second )
127       ++count;
128   }
129
130   return count;
131 }
132
133
134 template< typename JobInfoT, typename NodeInfoT >
135 inline
136 int
137 JobMap<JobInfoT, NodeInfoT>::get_free_node() const {
138   BOOST_FOREACH( const nodes_t::value_type &entry, nodes_ ) {
139     if( !entry.second )
140       return entry.first;
141   }
142
143   BOOST_THROW_EXCEPTION( AssertionError() );
144 }
145
146
147 template< typename JobInfoT, typename NodeInfoT >
148 inline
149 std::vector<Job::ptr_t>
150 JobMap<JobInfoT, NodeInfoT>::jobs() const {
151   std::vector<Job::ptr_t> jobs;
152
153   BOOST_FOREACH( const typename jobs_t::value_type &job, jobs_ ) {
154     jobs.push_back( job.first );
155   }
156
157   return jobs;
158 }
159
160
161 template< typename JobInfoT, typename NodeInfoT >
162 inline
163 Job::ptr_t
164 JobMap<JobInfoT, NodeInfoT>::job( int node ) const {
165   typename nodes_t::const_iterator node_it;
166   if( (node_it = nodes_.find(node)) == nodes_.end() )
167     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<int>(node
    ) );
168
169   if( !node_it->second )
170     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<int>(node
    ) );
171
172   return node_it->second;
173 }
174
175
176 template< typename JobInfoT, typename NodeInfoT >
177 inline
178 int
179 JobMap<JobInfoT, NodeInfoT>::optimizer_node( const Job::ptr_t &job )
    const {
180   typename jobs_t::const_iterator job_it;
181   if( (job_it = jobs_.find(job)) == jobs_.end() )
182     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<job_id_t
    >(job->id) );
183
184   return job_it->second.optimizer_node;
185 }
186
187
188 template< typename JobInfoT, typename NodeInfoT >
189 inline
190 std::vector<int>
191 JobMap<JobInfoT, NodeInfoT>::simulator_nodes( const Job::ptr_t &job )
    const {
192   std::vector<int> nodes;
193
194   typename jobs_t::const_iterator job_it;
195   if( (job_it = jobs_.find(job)) == jobs_.end() )
196     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<job_id_t
    >(job->id) );
197
198   BOOST_FOREACH( const typename simulator_nodes_t::value_type &entry,
    job_it->second.simulator_nodes ) {
199     nodes.push_back( entry.first );
200   }
201
202   return nodes;
203 }
204
```

```
205
206 template< typename JobInfoT, typename NodeInfoT >
207 inline
208 const JobInfoT &
209 JobMap<JobInfoT, NodeInfoT>::job_info( const Job::ptr_t &job ) const {
210   typename jobs_t::const_iterator job_it;
211   if( (job_it = jobs_.find(job)) == jobs_.end() )
212     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<job_id_t >(job->id) );
213
214   return job_it->second.job_info;
215 }
216
217
218 template< typename JobInfoT, typename NodeInfoT >
219 inline
220 JobInfoT &
221 JobMap<JobInfoT, NodeInfoT>::job_info( const Job::ptr_t &job ) {
222   typename jobs_t::iterator job_it;
223   if( (job_it = jobs_.find(job)) == jobs_.end() )
224     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<job_id_t >(job->id) );
225
226   return job_it->second.job_info;
227 }
228
229
230 template< typename JobInfoT, typename NodeInfoT >
231 inline
232 const NodeInfoT &
233 JobMap<JobInfoT, NodeInfoT>::node_info( const Job::ptr_t &job, int >(simulator_node ) const {
234   typename jobs_t::const_iterator job_it;
235   if( (job_it = jobs_.find(job)) == jobs_.end() )
236     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<job_id_t >(job->id) );
237
238   typename simulator_nodes_t::const_iterator sim_node_it;
239   if( (sim_node_it = job_it->second.simulator_nodes.find( >(simulator_node)) == job_it->second.simulator_nodes.end() )
240     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<int>( >(simulator_node) );
241
242   return sim_node_it->second.node_info;
243 }
244
245
246 template< typename JobInfoT, typename NodeInfoT >
247 inline
248 NodeInfoT &
249 JobMap<JobInfoT, NodeInfoT>::node_info( const Job::ptr_t &job, int >(simulator_node ) {
250   typename jobs_t::iterator job_it;
251   if( (job_it = jobs_.find(job)) == jobs_.end() )
252     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<job_id_t >(job->id) );
253
254   typename simulator_nodes_t::iterator sim_node_it;
255   if( (sim_node_it = job_it->second.simulator_nodes.find( >(simulator_node)) == job_it->second.simulator_nodes.end() )
256     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<int>( >(simulator_node) );
257
258   return sim_node_it->second.node_info;
259 }
```

Listing B.17: src/job_queue.hpp

```
1  #ifndef DICON_JOB_QUEUE_HPP_
2  #define DICON_JOB_QUEUE_HPP_
3
4  /*--- [Distribution] ---------------------------------------------------------
5   * This file is part of the Disease Control System DiCon.
6   *
7   * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8   * Designed and developed with the guidance of Nedialko B. Dimitrov
9   * and Lauren Ancel Meyers at the University of Texas at Austin.
10  *
11  * DiCon is free software: you can redistribute it and/or modify it
12  * under the terms of the GNU General Public License as published by
13  * the Free Software Foundation, either version 3 of the License, or
14  * (at your option) any later version.
15  *
16  * DiCon is distributed in the hope that it will be useful, but
17  * WITHOUT ANY WARRANTY; without even the implied warranty of
18  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19  * General Public License for more details.
20  *
21  * You should have received a copy of the GNU General Public License
22  * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23  *-----------------------------------------------------------------------------*/
24
25 /**
26  * @file
27  * @brief JobQueue class.
28  */
29 #include "error.hpp"
30 #include "job.hpp"
31 #include "lazy/set.hpp"
32 #include <boost/noncopyable.hpp>
33 #include <queue>
34
35
36 /// %Job queue related error.
37 struct JobQueueError : virtual Error
38 { virtual const char *what() const throw() { return "Job_queue_related_error."; } };
39
40 /// %Job queue has too many job.
41 struct JobQueueTooManyJobsError : virtual JobQueueError
42 { virtual const char *what() const throw() { return "Job_queue_has_too_many_jobs."; } };
43
44 /// %Job queue has not enough jobs.
45 struct JobQueueNotEnoughJobsError : virtual JobQueueError
46 { virtual const char *what() const throw() { return "Job_queue_has_not_enough_jobs."; } };
47
48
```

```
49  class Configuration;
50
51  /**
52   * @brief Lazy queue for jobs.
53   *
54   * The JobQueue class implements a lazy queue for jobs. This is the
55   * main class where new jobs (Job structure) are created. In order to
56   * do this in a memory-efficient way, new jobs are only created when
57   * requested.
58   *
59   * The job queue reads in a configuration and stores all the relevant
60   * information needed to create the jobs defined in the configuration
61   * at a later time.
62   *
63   * The methods empty(), front(), and pop() are used to iterate over
64   * the elements in the job queue. This can be accomplished as follows.
65   *
66   * @code
67  Configuration config;
68  // Read in configuration.
69
70  for( JobQueue queue(config); !queue.empty(); queue.pop() ) {
71    // Access job with queue.front().
72  }
73  @endcode
74   */
75  class JobQueue
76    : boost::noncopyable
77  {
78  public:
79    /**
80     * @brief Read in configuration.
81     *
82     * Constructor that reads in the configuration given by @e
83     * configuration and stores all the relevant information needed to
84     * create the jobs defined in that configuration at a later time.
85     *
86     * @param configuration %Configuration.
87     * @throws ConfigError or a derived class when necessary information
88     *   is missing from the configuration, or a config value could not
89     *   be interpreted correctly.
90     * @throws JobQueueTooManyJobsError when too many jobs have been
91     *   defined.
92     * @throws JobQueueNotEnoughJobsError when not enough jobs have been
93     *   defined.
94     */
95    JobQueue( const Configuration &configuration );
96
97  public:
98    /**
99     * @brief Remove front job from queue.
100    *
101    * Remove the front job from the job queue. This is used in
102    * combination with empty() and front() to iterate over all jobs.
103    *
104    * @throws AssertionError when job queue is empty.
105    */
106   void pop();
107
108 public:
109   /**
110    * @brief Check if job queue is empty.
111    *
112    * Check if the job queue is empty. This returns @c false if and
113    * only if there is at least one more job in the queue.
114    *
115    * @returns @c true iff job queue is empty.
116    */
117   bool empty() const;
118   /**
119    * @brief Get front job from queue.
120    *
121    * Get the front job from the job queue. This method creates the
122    * job (Job structure) as necessary on-demand. Subsequent calls to
123    * this method return the same job object until pop() is called.
124    *
125    * @returns Front job.
126    * @throws AssertionError when job queue is empty.
127    */
128   Job::ptr_t front() const;
129
130 public:
131   /**
132    * @brief Get total number of jobs.
133    *
134    * Get the total number of jobs in the queue. This includes all jobs
135    * already removed from the queue by pop().
136    *
137    * @returns Total number of jobs in queue.
138    */
139   size_t total_jobs() const;
140   /**
141    * @brief Get number of jobs done.
142    *
143    * Get the number of jobs that have already been removed from the
144    * queue by pop().
145    *
146    * @returns Number of jobs removed from queue.
147    */
148   size_t total_done() const;
149
150 private:
151   boost::filesystem::path job_dir( size_t job_set_id ) const;
152   boost::filesystem::path arg_dir( size_t job_set_id ) const;
153
154 private:
155   typedef LazySet<std::pair<arguments_t, arguments_t> > argument_set_t
      ;
156
157   struct JobTemplate {
158     Job::ptr_t base_job;
159     boost::shared_ptr<argument_set_t> argument_set;
160   };
161
162   std::queue<JobTemplate> jobs_;
163   mutable Job::ptr_t front_job_;
164
165   std::string job_dir_mask_;
166   std::string arg_dir_mask_;
167
168   std::string job_logfile_;
169   LogLevel    job_log_level_;
```

```
170     std::string checkpoint_file_;
171     std::string opt_logfile_mask_;
172     std::string sim_logfile_mask_;
173
174     std::string optimizer_map_file_mask_;
175     std::string optimizer_lib_file_mask_;
176     std::string policy_bin_dumpfile_mask_;
177     std::string policy_txt_dumpfile_mask_;
178
179     unsigned    policy_count_;
180     std::string policy_bin_result_;
181     std::string policy_txt_result_;
182
183     size_t job_count_;
184     size_t jobs_done_;
185 };
186
187 #endif //DICON_JOB_QUEUE_HPP_
```

Listing B.18: src/job_queue.cpp

```
1   /*---- [Distribution] ------------------------------------------------
2    * This file is part of the Disease Control System DiCon.
3    *
4    * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
5    * Designed and developed with the guidance of Nedialko B. Dimitrov
6    * and Lauren Ancel Meyers at the University of Texas at Austin.
7    *
8    * DiCon is free software: you can redistribute it and/or modify it
9    * under the terms of the GNU General Public License as published by
10   * the Free Software Foundation, either version 3 of the License, or
11   * (at your option) any later version.
12   *
13   * DiCon is distributed in the hope that it will be useful, but
14   * WITHOUT ANY WARRANTY; without even the implied warranty of
15   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
16   * General Public License for more details.
17   *
18   * You should have received a copy of the GNU General Public License
19   * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
20   *--------------------------------------------------------------------*/
21
22  #include "job_queue.hpp"
23  #include "specifier.hpp"
24  #include "config.hpp"
25  #include "lazy/set/combine.hpp"
26  #include "lazy/set/empty.hpp"
27  #include "lazy/set/singleton.hpp"
28  #include "message.hpp"
29  #include <boost/foreach.hpp>
30  #include <boost/format.hpp>
31  #include <cmath>
32
33
34  namespace detail {
35
36    static const size_t      default_max_jobs          = 1000000;
37
38    static const std::string default_job_dir_mask      = "job-%u";
39    static const std::string default_arg_dir_mask      = "argument-%05u";
40
41    static const std::string default_job_logfile       = "job.log";
42    static const LogLevel    default_job_log_level     = LOG_INFO;
43    static const std::string default_checkpoint_file   = "checkpoint.xml"
      ;
44    static const std::string default_opt_logfile_mask = "log/optimizer-
      node_%04u.log";
45    static const std::string default_sim_logfile_mask = "log/simulator-
      node_%04u.log";
46
47    static const std::string default_optimizer_map_file_mask  = "dump
      /%012u-optmap.bin";
48    static const std::string default_optimizer_lib_file_mask  = "dump
      /%012u-optlib.bin";
49    static const std::string default_policy_bin_dumpfile_mask = "dump
      /%012u-policy.bin";
50    static const std::string default_policy_txt_dumpfile_mask = "dump
      /%012u-policy.txt";
51
52    static const unsigned    default_policy_count      = 10;
53    static const std::string default_policy_bin_result = "result-policy.
      bin";
54    static const std::string default_policy_txt_result = "result-policy.
      txt";
55
56    static const simcount_t  default_max_sims          = 0;
57    static const unsigned    default_max_nodes         = 0;
58    static const unsigned    default_job_backlog       = 1;
59    static const unsigned    default_node_backlog      = 1;
60
61    static const unsigned    default_checkpoint_secs  = 14400;  // 4
      hours
62    static const unsigned    default_checkpoint_sims  = 0;
63
64
65    template< typename C, typename T = typename C::value_type >
66    struct PushBack {
67      typedef C result_type;
68
69      result_type
70      operator()( C x, const T &y ) const {
71        x.push_back( y );
72        return x;
73      }
74    };
75
76
77    template< typename T1, typename T2 >
78    struct MakePair {
79      typedef typename std::pair<T1, T2> result_type;
80
81      result_type
82      operator()( const T1 &x, const T2 &y ) const {
83        return std::make_pair<T1, T2>( x, y );
84      }
85    };
86
87
88    static
89    LazySet<arguments_t>::ptr_t
```

```
90    create_argument_set( const Configuration &configuration,
91                         const std::string &section, const std::string &parameter_mask )
92    {
93      LazySet<arguments_t>::ptr_t args
94        = lazy_singleton( arguments_t() );
95
96      std::string parameter;
97      for( unsigned parameter_id = 1
98         ; parameter = (boost::format(parameter_mask)
99                        % parameter_id).str()
100        , configuration.has( section, parameter )
101        ; ++parameter_id
102        )
103     {
104       try {
105         const std::string &argument = configuration.get<std::string>( section, parameter );
106
107         args = lazy_combine( args, parse_argument_specifier(argument), detail::PushBack<arguments_t>() );
108       }
109       catch( boost::exception &e ) {
110         e << errinfo_config_parameter(parameter);
111         throw;
112       }
113     }
114
115     return args;
116   }
117
118
119   static
120   LazySet<simcount_t>::ptr_t
121   create_checkpoints( const Configuration &configuration,
122                       const std::string &section, const std::string &parameter )
123   {
124     try {
125       const std::string &checkpoints = configuration.get<std::string>( section, parameter, std::string() );
126
127       if( checkpoints.empty() )
128         return lazy_empty<simcount_t>();
129       else
130         return parse_schedule_specifier(checkpoints);
131     }
132     catch( boost::exception &e ) {
133       e << errinfo_config_parameter(parameter);
134       throw;
135     }
136   }
137
138 }
139
140
141
142 JobQueue::JobQueue( const Configuration &configuration )
143   : job_count_(0), jobs_done_(0)
144 {
145   size_t max_jobs = configuration.get<size_t>( "global", "max_jobs", detail::default_max_jobs );
146
147   job_dir_mask_     = configuration.get<std::string>( "global", "job_dir"       , detail::default_job_dir_mask     );
148   arg_dir_mask_     = configuration.get<std::string>( "global", "arg_dir"       , detail::default_arg_dir_mask     );
149   job_logfile_      = configuration.get<std::string>( "global", "job_logfile"   , detail::default_job_logfile      );
150   job_log_level_    = configuration.get<LogLevel   >( "global", "job_log_level", detail::default_job_log_level    );
151   checkpoint_file_  = configuration.get<std::string>( "global", "checkpoint"    , detail::default_checkpoint_file  );
152   opt_logfile_mask_ = configuration.get<std::string>( "global", "opt_logfile"   , detail::default_opt_logfile_mask );
153   sim_logfile_mask_ = configuration.get<std::string>( "global", "sim_logfile"   , detail::default_sim_logfile_mask );
154
155   optimizer_map_file_mask_  = configuration.get<std::string>( "global", "optimizer_map_file",
156                                                               detail::default_optimizer_map_file_mask  );
157   optimizer_lib_file_mask_  = configuration.get<std::string>( "global", "optimizer_lib_file",
158                                                               detail::default_optimizer_lib_file_mask  );
159   policy_bin_dumpfile_mask_ = configuration.get<std::string>( "global", "policy_bin_dumpfile",
160                                                               detail::default_policy_bin_dumpfile_mask );
161   policy_txt_dumpfile_mask_ = configuration.get<std::string>( "global", "policy_txt_dumpfile",
162                                                               detail::default_policy_txt_dumpfile_mask );
163
164   policy_count_      = configuration.get<unsigned   >( "global", "policy_count"      , detail::default_policy_count      );
165   policy_bin_result_ = configuration.get<std::string>( "global", "policy_bin_result", detail::default_policy_bin_result );
166   policy_txt_result_ = configuration.get<std::string>( "global", "policy_txt_result", detail::default_policy_txt_result );
167
168   std::string section;
169   for( unsigned section_id = 1
170      ; section = (boost::format("job-%u") % section_id).str()
171      , configuration.has( section )
172      ; ++section_id
173      )
174   {
175     try {
176       Job::ptr_t base_job( new Job() );
177
178       base_job->job_set_id        = section_id;
179       base_job->arg_set_id        = 0;
180
181       base_job->max_sims          = configuration.get<simcount_t >( section, "max_sims"       , detail::default_max_sims        );
182       base_job->max_nodes         = configuration.get<unsigned   >( section, "max_nodes"      , detail::default_max_nodes       );
183       base_job->job_backlog       = configuration.get<unsigned   >( section, "job_backlog"    , detail::default_job_backlog     );
```

```
184       base_job->node_backlog      = configuration.get<unsigned   >( section, "node_backlog"    , detail::default_node_backlog    );
185
186       base_job->checkpoint_secs   = configuration.get<unsigned   >( section, "checkpoint_secs", detail::default_checkpoint_secs );
187       base_job->checkpoint_sims   = configuration.get<unsigned   >( section, "checkpoint_sims", detail::default_checkpoint_sims );
188
189       base_job->checkpoints.reset( detail::create_checkpoints( configuration, section, "checkpoints").release() );
190
191       base_job->optimizer_library = configuration.get<std::string>( section, "optimizer" );
192       base_job->simulator_command = configuration.get<std::string>( section, "simulator" );
193
194       LazySet<arguments_t>::ptr_t opt_args
195         = detail::create_argument_set( configuration, section, "opt_args[%u]" );
196       LazySet<arguments_t>::ptr_t sim_args
197         = detail::create_argument_set( configuration, section, "sim_args[%u]" );
198
199       assert( opt_args.get() );
200       assert( sim_args.get() );
201
202       argument_set_t::ptr_t argument_set
203         = lazy_combine( opt_args, sim_args, detail::MakePair<arguments_t, arguments_t>() );
204
205       size_t job_count = 0;
206
207       // Make sure number of jobs does not exceed limit.
208       for( argument_set->reset(); argument_set->has(); argument_set->inc(), ++job_count ) {
209         if( (job_count_ + job_count) >= max_jobs )
210           BOOST_THROW_EXCEPTION( JobQueueTooManyJobsError() << errinfo_max_value<size_t>(max_jobs) );
211       }
212
213       if( job_count == 0 )
214         BOOST_THROW_EXCEPTION( JobQueueNotEnoughJobsError() << errinfo_min_value<size_t>(1) );
215
216       JobTemplate job_tmpl;
217
218       argument_set->reset();
219       job_tmpl.base_job = base_job;
220       job_tmpl.argument_set.reset( argument_set.release() );
221
222       job_count_ += job_count;
223       jobs_.push( job_tmpl );
224     }
225     catch( boost::exception &e ) {
226       e << errinfo_config_section(section);
227       throw;
228     }
229   }
230 }
231
232
```

```
233 void
234 JobQueue::pop() {
235   if( jobs_.empty() )
236     BOOST_THROW_EXCEPTION( AssertionError() );
237
238   jobs_.front().argument_set->inc();
239
240   if( !jobs_.front().argument_set->has() ) {
241     // Go to next job/ argument set.
242     jobs_.pop();
243   }
244
245   front_job_.reset();
246
247   ++jobs_done_;
248 }
249
250
251 bool
252 JobQueue::empty() const {
253   bool res = jobs_.empty();
254   assert( bool(jobs_done_ == job_count_) == res );
255   return res;
256 }
257
258
259 Job::ptr_t
260 JobQueue::front() const {
261   if( jobs_.empty() )
262     BOOST_THROW_EXCEPTION( AssertionError() );
263
264   if( !front_job_ ) {
265     // Increment the job's argument set id.
266     ++(jobs_.front().base_job->arg_set_id);
267
268     Job::ptr_t job( new Job(*jobs_.front().base_job) );
269
270     const argument_set_t::value_t &arguments
271       = jobs_.front().argument_set->get();
272
273     job->id               = job_id_t( jobs_done_+1 );
274
275     assert( job->checkpoints );
276     // Call an explicit clone() on checkpoint lazy set to break sharing.
277     job->checkpoints.reset( job->checkpoints->clone().release() );
278     job->checkpoints->reset();
279
280     job->job_directory    = job_dir(job->job_set_id) / arg_dir(job->arg_set_id);
281
282     job->job_logfile      = job_logfile_;
283     job->job_log_level    = job_log_level_;
284     job->checkpoint_file  = checkpoint_file_;
285
286     job->opt_logfile_mask = opt_logfile_mask_;
287     job->sim_logfile_mask = sim_logfile_mask_;
288
289     job->optimizer_map_file_mask  = optimizer_map_file_mask_;
290     job->optimizer_lib_file_mask  = optimizer_lib_file_mask_;
291     job->policy_bin_dumpfile_mask = policy_bin_dumpfile_mask_;
```

```
292     job->policy_txt_dumpfile_mask = policy_txt_dumpfile_mask_;
293
294     job->policy_count      = policy_count_;
295     job->policy_bin_result = policy_bin_result_;
296     job->policy_txt_result = policy_txt_result_;
297
298     job->optimizer_arguments = arguments.first;
299     job->simulator_arguments = arguments.second;
300
301     front_job_ = job;
302   }
303
304   return front_job_;
305 }
306
307
308 size_t
309 JobQueue::total_jobs() const {
310   return job_count_;
311 }
312
313
314 size_t
315 JobQueue::total_done() const {
316   return jobs_done_;
317 }
318
319
320 boost::filesystem::path
321 JobQueue::job_dir( size_t job_set_id ) const {
322   return (boost::format(job_dir_mask_) % job_set_id).str();
323 }
324
325
326 boost::filesystem::path
327 JobQueue::arg_dir( size_t arg_set_id ) const {
328   return (boost::format(arg_dir_mask_) % arg_set_id).str();
329 }
```

Listing B.19: src/library.hpp

```
1  #ifndef DICON_LIBRARY_HPP_
2  #define DICON_LIBRARY_HPP_
3
4  /*--- [Distribution] ------------------------------------------------
5   * This file is part of the Disease Control System DiCon.
6   *
7   * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8   * Designed and developed with the guidance of Nedialko B. Dimitrov
9   * and Lauren Ancel Meyers at the University of Texas at Austin.
10  *
11  * DiCon is free software: you can redistribute it and/or modify it
12  * under the terms of the GNU General Public License as published by
13  * the Free Software Foundation, either version 3 of the License, or
14  * (at your option) any later version.
15  *
16  * DiCon is distributed in the hope that it will be useful, but
17  * WITHOUT ANY WARRANTY; without even the implied warranty of
18  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19  * General Public License for more details.
20  *
21  * You should have received a copy of the GNU General Public License
22  * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23  *-----------------------------------------------------------------*/
24
25 #include "error.hpp"
26 #include <boost/noncopyable.hpp>
27
28
29 /// Errinfo storing the dlerror() %message.
30 DICON_ERRINFO( library_dlerror, std::string );
31 /// Errinfo storing the library symbol name.
32 DICON_ERRINFO( library_symbol, std::string );
33
34
35 /// Dynamic-link library related error.
36 struct LibraryError : virtual SystemError
37 { virtual const char *what() const throw() { return "Dynamic-link_
  library_related_error."; } };
38
39 /// Failed to open dynamic-link library.
40 struct LibraryOpenError : virtual LibraryError
41 { virtual const char *what() const throw() { return "Failed_to_open_
  dynamic-link_library."; } };
42
43 /// Symbol not found in dynamic-link library.
44 struct LibrarySymbolError : virtual LibraryError
45 { virtual const char *what() const throw() { return "Symbol_not_found_
  in_dynamic-link_library."; } };
46
47
48 /**
49  * @brief Dynamic-link library access.
50  *
51  * The DynamicLibrary class provides access to the symbols exported by
52  * a dynamic-link library (DLL).  Exceptions are thrown when the
53  * library cannot be opened or a specific symbol cannot be found.
54  *
55  * As the symbols returned by operator[]() are @c void pointers, an
56  * explicit @c reinterpret_cast to the specific type is needed to make
57  * use of them. The following code demonstrates this for functions by
58  * opening a dynamic library, getting a symbol representing a function
59  * with a single @c int parameter and returning an @c int, and calling
60  * this function.  Non-function symbols can be accessed in a similar
61  * way.
62  *
63  * @code
64 // Open the dynamic library "some_library.so".
65 DynamicLibrary library( "some_library.so" );
66
67 // In order to load exported functions, an explicit cast to the
68 // function pointer type is required. Define this type here.
69 typedef int (*some_function_t)( int some_parameter );
70
71 // Now get the symbol and cast it to the expected function type.
72 some_function_t some_function =
73   reinterpret_cast<some_function_t>( library["some_function"] );
74
75 // Finally, call the function defined in the dynamic library.
76 std::cout << some_function(42) << std::endl;
```

```
77  @endcode
78   */
79  class DynamicLibrary
80    : boost::noncopyable
81  {
82  public:
83    /**
84     * @brief Open dynamic-link library.
85     *
86     * Constructor that opens the dynamic-link library given by @c file.
87     *
88     * @param file Filename of the dynamic library.
89     * @throws LibraryOpenError when the library cannot be opened.
90     */
91    DynamicLibrary( const std::string &file );
92    /**
93     * @brief Close library and relase resources.
94     *
95     * Destructor that closes the library and releases all associated
96     * resources.
97     */
98    ~DynamicLibrary();
99
100 public:
101   /**
102    * @brief Get exported symbol from library.
103    *
104    * Get a symbol that is exported by the dynamic-link library. An
105    * exception is thrown when the symbol cannot be found, i.e., when
106    * the library does not export the given symbol. As the symbol is
107    * returned through a @c void pointer, an explicit @c
108    * reinterpret_cast to the specific type is needed to make use of
109    * it. See the description of the DynamicLibrary class for an
110    * example on how to do this.
111    *
112    * @param symbol Name of the symbol.
113    * @returns @c void pointer to the symbol given by @c symbol.
114    * @throws LibrarySymbolError when the symbol cannot be found.
115    */
116   void *operator[]( const std::string &symbol );
117
118 private:
119   void *handle_;
120 };
121
122 #endif //DICON_LIBRARY_HPP_
```

Listing B.20: src/library.cpp

```
1   /*---- [Distribution] ------------------------------------------------
2    * This file is part of the Disease Control System DiCon.
3    *
4    * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
5    * Designed and developed with the guidance of Nedialko B. Dimitrov
6    * and Lauren Ancel Meyers at the University of Texas at Austin.
7    *
8    * DiCon is free software: you can redistribute it and/or modify it
9    * under the terms of the GNU General Public License as published by
10   * the Free Software Foundation, either version 3 of the License, or
11   * (at your option) any later version.
12   *
13   * DiCon is distributed in the hope that it will be useful, but
14   * WITHOUT ANY WARRANTY; without even the implied warranty of
15   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
16   * General Public License for more details.
17   *
18   * You should have received a copy of the GNU General Public License
19   * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
20   *--------------------------------------------------------------------*/
21
22  #include "library.hpp"
23  #include "file.hpp"
24  #include <boost/format.hpp>
25  #include <dlfcn.h>
26
27
28  namespace detail {
29
30    std::string
31    dlerror() {
32      char *res;
33
34      if( (res = ::dlerror()) != NULL )
35        return res;
36      else
37        return "NULL";
38    }
39
40  }
41
42
43
44  DynamicLibrary::DynamicLibrary( const std::string &file ) {
45    if( (handle_ = dlopen(file.c_str(), RTLD_NOW)) == NULL ) {
46      BOOST_THROW_EXCEPTION( LibraryOpenError() << errinfo_api_function(
    "dlopen")
47                             << errinfo_file_name(file) <<
    errinfo_library_dlerror(detail::dlerror()) );
48    }
49  }
50
51
52  DynamicLibrary::~DynamicLibrary() {
53    dlclose( handle_ );
54    // Ignore errors.
55  }
56
57
58  void *
59  DynamicLibrary::operator[]( const std::string &symbol ) {
60    void *res;
61
62    if( (res = dlsym(handle_, symbol.c_str())) == NULL ) {
63      BOOST_THROW_EXCEPTION( LibrarySymbolError() <<
    errinfo_api_function("dlsym")
64                             << errinfo_library_symbol(symbol) <<
    errinfo_library_dlerror(detail::dlerror()) );
65    }
66
67    return res;
```

```
}
```

Listing B.21: src/log.hpp

```
#ifndef DICON_LOG_HPP_
#define DICON_LOG_HPP_

/*---- [Distribution] ------------------------------------------------
 * This file is part of the Disease Control System DiCon.
 *
 * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
 * Designed and developed with the guidance of Nedialko B. Dimitrov
 * and Lauren Ancel Meyers at the University of Texas at Austin.
 *
 * DiCon is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * DiCon is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
 *--------------------------------------------------------------------*/

/**
 * @file
 * @brief Log related classes and functions.
 *
 * The log.hpp file provides the global log system. Log sinks can be
 * created by deriving from LogSink, or instantiating the LogFile
 * class. Sinks are registered with the global log system using
 * log_register_sink(), and are removed from the global log system by
 * log_remove_sink(). Log sinks are being held by weak pointers:
 * therefore, they can be removed at arbitrary times by simply
 * destroying all remaining smart pointers to the sink; calling
 * log_remove_sink() is not necessary.
 *
 * Log messages are created in various log levels using the global
 * functions log_debug(), log_info(), log_warning(), log_error(), and
 * log_failure(). Each of these functions is guaranteed not to throw
 * an exception. The %message is automatically distributed to all
 * registered log sinks. Messages of the error and failure log levels
 * are also written to standard error output. If writing a log
 * %message to a sink fails, that sink will be removed from the global
 * log system and an error %message will be written to standard error
 * output.
 */
#include "error.hpp"
#include "types.hpp"
#include <boost/filesystem/path.hpp>
#include <boost/noncopyable.hpp>
#include <boost/shared_ptr.hpp>
#include <fstream>
#include <string>

/// Log related error.
struct LogError : virtual Error
{ virtual const char *what() const throw() { return "Log_related_error.
"; } };

/// Writing to log failed.
struct LogWriteError : public virtual IOWriteError, public virtual
 LogError
{ virtual const char *what() const throw() { return "Writing_to_log_
failed."; } };


/**
 * @brief Sink used for logging.
 *
 * The LogSink class is the base class for sinks used for logging. A
 * log sink can be an arbitrary target for log messages, such as a
 * logfile (LogFile class).
 */
class LogSink {
public:
  virtual ~LogSink() {}

public:
  /**
   * @brief Write log %message to sink.
   *
   * Write the log %message given by @e %message with log level given
   * by @e level to the log sink.
   *
   * @param message Log %message.
   * @param level Log level.
   */
  virtual void log( const std::string &message, LogLevel level ) = 0;
};


/**
 * @brief Logfile sink used for logging.
 *
 * The LogFile class is a log sink that writes log messages to
 * file. The messages are prefixed with the current date and time, and
 * a textual representation of the log level of the %message. Also,
 * messages containing line breaks are spread over multiple lines with
 * the second and following lines indented. The following is an
 * example how the resulting logfile will look like.
 *
 * @verbatim
Nov 05 04:45:44 info: Starting job (#5; job-1, argument-5) in
 directory 'job-1/argument-00005'.
Nov 05 04:45:44 info|    Optimizer command line: '../optimizer/
exhaustive.so'
Nov 05 04:45:44 info|    Simulator command line: '../simulator/python/
simple_simulator.py 5 5'
Nov 05 04:45:44 info: Checkpoint file 'job-1/argument-00005/checkpoint
.xml' does not exist.
Nov 05 04:45:44 info: Job has no checkpoints; beginning job from
 scratch.
Nov 05 04:45:44 info: Initializing optimizer on node #135.
@endverbatim
```

```
 */
class LogFile
  : boost::noncopyable, public LogSink
{
public:
  /**
   * @brief Create new logfile.
   *
   * Constructor that creates the logfile given by @e file. If a file
   * with that name already exists, it is not overwritten but an
   * alternative filename that does not yet exist is chosen by the
   * means of File::unique().
   */
  LogFile( const boost::filesystem::path &file );
  /**
   * @brief Close logfile.
   *
   * Destructor that close the logfile. Before closing and flushing
   * the file to disk, this writes the log %message "Closing logfile."
   * to the end of the logfile. By the absence of this %message,
   * logfiles that have not been closed properly can be identified,
   * e.g., in the event of a system crash.
   */
  virtual ~LogFile();

public:
  /**
   * @brief Write log %message to logfile.
   *
   * Write the log %message given by @e %message with log level given
   * by @e level to the logfile. See the description of the LogFile
   * class for how the output format looks like.
   *
   * @param message Log %message.
   * @param level Log level.
   *
   * @throws LogWriteError when log %message cannot be written.
   */
  virtual void log( const std::string &message, LogLevel level );

private:
  std::string log_filename_;
  std::ofstream output_;
};


/**
 * @brief Register log sink with global log system.
 *
 * Register the log sink given by @e sink with the global log
 * system. Only log messages with log level of at least @e min_level
 * are sent to this sink. If @e job_id is not equal to 0, only
 * messages regarding the job given by @e job_id are sent to this
 * sink. If @e job_id is equal to 0, all messages are sent to this
 * sink, with a prefix of <code>[%Job \#X]</code>, where @c X is the
 * job number, in case the %message is connected to a job (no prefix
 * is added for log messages that are not connected to a job).
 *
 * @note It is not an error to register a log sink more than once. In
 * this case, log messages are sent to the same log sink multiple
 * times, once for each registration (given that they match each
 * registration's filter conditions defined by @e min_level and @e
 * job_id). In case of an error only one of the registrations is
 * removed automatically per log %message (see the description of the
 * log.hpp header for more information).
 *
 * @param sink Log sink to register.
 * @param min_level Minimum log level for this sink.
 * @param job_id %Job ID to log (or 0 for all job IDs).
 */
void log_register_sink( const boost::shared_ptr<LogSink> &sink,
LogLevel min_level, job_id_t job_id = 0 );
/**
 * @brief Remove log sink from global log system.
 *
 * Remove the log sink given by @e sink from the global log system. If
 * the sink has been registered more than once, all registrations are
 * removed.
 *
 * @param sink Log sink to remove.
 */
void log_remove_sink( const boost::shared_ptr<LogSink> &sink );

/**
 * @brief Send debug log %message.
 *
 * Send the log %message given by @e %message with the debug log level
 * to the global log system. If @e job_id is equal to 0, the %message
 * is sent to all log sinks that had @e job_id set to 0 when calling
 * log_register_sink(). If @e job_id is not equal to 0, the %message
 * is sent only to sinks registered with the same @e job_id, or sinks
 * registered with @e job_id set to 0 (in this case, a prefix of
 * <code>[%Job \#X]</code> is added to the %message, where @c X is the
 * job number given by @e job_id).
 *
 * This function is guaranteed not to throw an exception.
 *
 * @param message Log %message.
 * @param job_id Optional job ID.
 */
void log_debug   ( const std::string &message, job_id_t job_id = 0 )
throw();
/**
 * @brief Send info log %message.
 *
 * Send the log %message given by @e %message with the info log level
 * to the global log system.
 *
 * @see log_debug().
 */
void log_info    ( const std::string &message, job_id_t job_id = 0 )
throw();
/**
 * @brief Send warning log %message.
 *
 * Send the log %message given by @e %message with the warning log
 * level to the global log system.
 *
 * @see log_debug().
 */
void log_warning( const std::string &message, job_id_t job_id = 0 )
throw();
```

```
227 /**
228  * @brief Send error log %message.
229  *
230  * Send the log %message given by @e %message with the error log level
231  * to the global log system.
232  *
233  * @see log_debug().
234  */
235 void log_error   ( const std::string &message, job_id_t job_id = 0 )
    throw();
236 /**
237  * @brief Send failure log %message.
238  *
239  * Send the  log %message  given by @e  %message with the  failure log
240  * level to the global log system.
241  *
242  * @see log_debug().
243  */
244 void log_failure( const std::string &message, job_id_t job_id = 0 )
    throw();
245
246 /// Send debug log %message.
247 void log_debug   ( job_id_t job_id, const std::string &message ) throw
    ();
248 /// Send info log %message.
249 void log_info    ( job_id_t job_id, const std::string &message ) throw
    ();
250 /// Send warning log %message.
251 void log_warning( job_id_t job_id, const std::string &message ) throw
    ();
252 /// Send error log %message.
253 void log_error   ( job_id_t job_id, const std::string &message ) throw
    ();
254 /// Send failure log %message.
255 void log_failure( job_id_t job_id, const std::string &message ) throw
    ();
256
257 #endif //DICON_LOG_HPP_
```

Listing B.22: src/log.cpp

```
1  /*--- [Distribution] ------------------------------------------------
2   * This file is part of the Disease Control System DiCon.
3   *
4   * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
5   * Designed and developed with the guidance of Nedialko B. Dimitrov
6   * and Lauren Ancel Meyers at the University of Texas at Austin.
7   *
8   * DiCon is free software: you  can redistribute it and/or modify it
9   * under the terms of the GNU General Public License as published by
10  * the Free Software Foundation, either version 3 of the License, or
11  * (at your option) any later version.
12  *
13  * DiCon  is distributed in  the hope  that it  will be  useful, but
14  * WITHOUT  ANY  WARRANTY;  without  even the  implied  warranty  of
15  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
16  * General Public License for more details.
17  *
18  * You should have received a copy of the GNU General Public License
19  * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
20  *------------------------------------------------------------------*/
21
22 #include "log.hpp"
23 #include "file.hpp"
24 #include <boost/foreach.hpp>
25 #include <boost/format.hpp>
26 #include <boost/scoped_ptr.hpp>
27 #include <boost/version.hpp>
28 #include <boost/weak_ptr.hpp>
29 #include <iostream>
30 #include <vector>
31
32
33 namespace detail {
34
35   class LogManager
36     : boost::noncopyable
37   {
38   public:
39     void register_sink( const boost::shared_ptr<LogSink> &sink,
    LogLevel min_level, job_id_t job_id );
40     void remove_sink( const boost::shared_ptr<LogSink> &sink );
41
42   public:
43     void log( const std::string &message, LogLevel level, job_id_t
    job_id );
44
45   private:
46     struct SinkData {
47       boost::weak_ptr<LogSink> sink;
48       LogLevel min_level;
49       job_id_t job_id;
50
51       void swap( SinkData &other );
52     };
53
54     std::vector<SinkData> sinks_;
55   };
56
57 }
58
59
60
61 namespace detail {
62
63   static
64   std::string
65   with_job( const std::string &message, job_id_t job_id ) {
66     assert( job_id > 0 );
67     return (boost::format("[Job #%u] %s") % job_id % message).str();
68   }
69
70 }
71
72
73
74 namespace detail {
75
76   void
77   LogManager::SinkData::swap( SinkData &other ) {
```

```
78      sink.swap( other.sink );
79      std::swap( min_level, other.min_level );
80      std::swap( job_id, other.job_id );
81    }
82
83
84    void
85    LogManager::register_sink( const boost::shared_ptr<LogSink> &sink,
      LogLevel min_level, job_id_t job_id ) {
86      SinkData data;
87
88      data.sink = sink;
89      data.min_level = min_level;
90      data.job_id = job_id;
91
92      sinks_.push_back( data );
93    }
94
95
96    void
97    LogManager::remove_sink( const boost::shared_ptr<LogSink> &sink ) {
98      size_t i = 0;
99
100     while( i < sinks_.size() ) {
101       const SinkData &data = sinks_[i++];
102       boost::shared_ptr<LogSink> entry = data.sink.lock();
103
104       if( !entry || entry == sink ) {
105         // Remove the sink, then continue.
106         sinks_[--i].swap( sinks_.back() );
107         sinks_.pop_back();
108         continue;
109       }
110     }
111   }
112
113
114   void
115   LogManager::log( const std::string &message, LogLevel level,
      job_id_t job_id ) {
116     boost::exception_ptr error;
117
118     size_t i = 0;
119     while( i < sinks_.size() ) {
120       const SinkData &data = sinks_[i++];
121       boost::shared_ptr<LogSink> sink = data.sink.lock();
122
123       if( !sink ) {
124         // Remove the sink, then continue.
125         sinks_[--i].swap( sinks_.back() );
126         sinks_.pop_back();
127         continue;
128       }
129
130       bool job_matches = data.job_id == 0 || data.job_id == job_id;
131
132       if( !(job_matches && level >= data.min_level) ) {
133         // Do not log to this sink.
134         continue;
135       }
136
137       try {
138         if( job_id == 0 || data.job_id != 0 )
139           sink->log( message, level );
140         else
141           sink->log( with_job(message, job_id), level );
142       }
143       catch( ... ) {
144         if( !error ) {
145           // Keep 1st exception to be thrown.
146           error = boost::current_exception();
147
148           // Remove the sink, then continue.
149           sinks_[--i].swap( sinks_.back() );
150           sinks_.pop_back();
151           continue;
152         }
153       }
154     }
155
156     if( error )
157       boost::rethrow_exception( error );
158   }
159
160 }
161
162
163
164 namespace detail {
165
166   static const int stderr_clone = dup( STDERR_FILENO );
167   static const char time_format[] = "%b %d %H:%M:%S";
168   static LogManager log_manager;
169
170
171   static
172   std::string
173   level_str( LogLevel level ) {
174     switch( level ) {
175     case LOG_DEBUG:   return "debug";
176     case LOG_INFO:    return "info";
177     case LOG_WARNING: return "warning";
178     case LOG_ERROR:   return "error";
179     case LOG_FAILURE: return "failure";
180     }
181
182     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<LogLevel
      >(level) );
183   }
184
185
186   static
187   std::string
188   time_str() {
189     time_t current;
190     if( (current = time(NULL)) == time_t(-1) )
191       BOOST_THROW_EXCEPTION( TimeError() << errinfo_api_function("time
      ") << errinfo_errno(errno) );
192
193     struct tm tm_time;
194     if( localtime_r(&current, &tm_time) == NULL )
```

```
195       BOOST_THROW_EXCEPTION( TimeError() << errinfo_api_function("localtime_r") << errinfo_value<time_t>(current) );
196
197     static const size_t max_length = 128;
198
199     char result[ max_length ];
200     size_t length = strftime( result, max_length, time_format, &tm_time );
201
202     return std::string( result, length );
203   }
204
205
206   static
207   std::vector<std::string>
208   log_to_lines( const std::string &message, LogLevel level ) {
209     static const char line_separators[] = "\n\r";
210     static const char whitespace_chars[] = " \t";
211
212     std::vector<std::string> lines;
213
214     std::string prefix = (boost::format("%s %s") % time_str() % level_str(level)).str();
215
216     size_t p = 0, np;
217     do {
218       np = message.find_first_of( line_separators, p );
219       const std::string &line = message.substr( p, np != std::string::npos ? np - p : np );
220
221       if( line.find_first_not_of(whitespace_chars) != std::string::npos )
222         lines.push_back( prefix + (lines.empty() ? ": " : "|   ") + line );
223
224       p = message.find_first_not_of( line_separators, np );
225     } while( p != std::string::npos );
226
227     return lines;
228   }
229
230
231   static
232   void
233   log_write( int fd, const std::string &line, bool flush = true ) throw() {
234     try {
235       if( fd < 0 ) {
236         // Ignore.
237         return;
238       }
239
240       size_t done = 0;
241       while( done < line.length() ) {
242         ssize_t res;
243         if( (res = ::write(fd, line.c_str()+done, line.length()-done)) <= 0 ) {
244           // Ignore.
245           return;
246         }
247
248         assert( size_t(res) <= line.length()-done );
249
250         done += size_t(res);
251       }
252
253       if( flush )
254         ::fsync( fd );
255     }
256     catch( ... ) {
257       // Ignore.
258     }
259   }
260
261
262   static
263   void
264   log_message( const std::string &message, LogLevel level, job_id_t job_id ) throw() {
265     try {
266       boost::exception_ptr error;
267
268       // Output message on sinks as appropriate.
269
270       try {
271         log_manager.log( message, level, job_id );
272       }
273       catch( ... ) {
274         error = boost::current_exception();
275       }
276
277       // In addition, output errors to stderr output.
278
279       if( level >= LOG_ERROR ) {
280         try {
281           const std::string &msg = job_id == 0 ? message : with_job( message, job_id );
282           BOOST_FOREACH( const std::string &line, log_to_lines(msg, level) ) {
283             log_write( stderr_clone, line+'\n' );
284           }
285         }
286         catch( ... ) {
287           // Ignore.
288         }
289       }
290
291       // If message output on sink failed, print error.
292
293       if( error ) {
294         try {
295           std::string msg = "Failed to write message to log.\n";
296 #if BOOST_VERSION >= 104000
297           msg += boost::diagnostic_information( error );
298 #elif BOOST_VERSION >= 103900
299           try {
300             boost::rethrow_exception( error );
301           }
302           catch( ... ) {
303             msg += boost::current_exception_diagnostic_information();
304           }
305 #else
```

```
306             try {
307               boost::rethrow_exception( error );
308             }
309             catch( boost::exception &e ) {
310               msg += boost::diagnostic_information( e );
311             }
312             catch( ... ) {
313               msg += "<unknown exception>\n";
314             }
315 #endif
316             BOOST_FOREACH( const std::string &line, log_to_lines(msg,
    LOG_ERROR) ) {
317               log_write( stderr_clone, line+'\n' );
318             }
319           }
320           catch( ... ) {
321             // Ignore.
322           }
323         }
324       }
325       catch( ... ) {
326         // Ignore.
327       }
328     }
329
330 }
331
332
333 LogFile::LogFile( const boost::filesystem::path &file )
334   : log_filename_(File::unique_name(file).file_string()),
335     output_(log_filename_.c_str())
336 {
337 }
338
339
340 LogFile::~LogFile() {
341   try {
342     log( "Closing logfile.", LOG_INFO );
343     output_.close();
344   }
345   catch( ... ) {
346     // Ignore.
347   }
348 }
349
350
351 void
352 LogFile::log( const std::string &message, LogLevel level ) {
353   BOOST_FOREACH( const std::string &line, detail::log_to_lines(message
    , level) ) {
354     output_ << line << '\n';
355   }
356
357   output_ << std::flush;
358
359   if( output_.fail() )
360     BOOST_THROW_EXCEPTION( LogWriteError() << errinfo_file_name(
    log_filename_) );
361 }
362
363
```

```
364 void
365 log_register_sink( const boost::shared_ptr<LogSink> &sink, LogLevel
    min_level, job_id_t job_id ) {
366   detail::log_manager.register_sink( sink, min_level, job_id );
367 }
368
369
370 void
371 log_remove_sink( const boost::shared_ptr<LogSink> &sink ) {
372   detail::log_manager.remove_sink( sink );
373 }
374
375
376 #define DICON_DEFINE_LOGS_( NAME, LEVEL )                              
    \
377   void log_##NAME( const std::string &message,                         
    \
378                    job_id_t job_id ) throw()                           
    \
379   {                                                                    
    \
380     detail::log_message( message, LOG_##LEVEL, job_id );               
    \
381   }                                                                    
    \
382                                                                        
    \
383   void log_##NAME( job_id_t job_id,                                    
    \
384                    const std::string &message ) throw()                
    \
385   {                                                                    
    \
386     assert( job_id != 0 );                                             
    \
387     log_##NAME( message, job_id );                                     
    \
388   }                                                                    
    \
389   /**/
390
391 DICON_DEFINE_LOGS_( debug  , DEBUG   )
392 DICON_DEFINE_LOGS_( info   , INFO    )
393 DICON_DEFINE_LOGS_( warning, WARNING )
394 DICON_DEFINE_LOGS_( error  , ERROR   )
395 DICON_DEFINE_LOGS_( failure, FAILURE )
396
397 #undef DICON_DEFINE_LOGS_
```

Listing B.23: src/main.hpp

```
1 #ifndef DICON_MAIN_HPP_
2 #define DICON_MAIN_HPP_
3
4 /*--- [Distribution] ------------------------------------------------
5  * This file is part of the Disease Control System DiCon.
6  *
7  * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8  * Designed and developed with the guidance of Nedialko B. Dimitrov
```

```
 9  * and Lauren Ancel Meyers at the University of Texas at Austin.
10  *
11  * DiCon is free software: you can redistribute it and/or modify it
12  * under the terms of the GNU General Public License as published by
13  * the Free Software Foundation, either version 3 of the License, or
14  * (at your option) any later version.
15  *
16  * DiCon is distributed in the hope that it will be useful, but
17  * WITHOUT ANY WARRANTY; without even the implied warranty of
18  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
19  * General Public License for more details.
20  *
21  * You should have received a copy of the GNU General Public License
22  * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
23  *----------------------------------------------------------------*/
24
25 /**
26  * @file
27  * @brief The main() function.
28  */
29 /**
30  * @mainpage
31  *
32  * DiCon is the Disease Control System. It provides a general
33  * framework for solving optimization problems on distributed computer
34  * clusters.
35  *
36  * This is the main source documentation on DiCon. It is meant to
37  * provide an introduction to how the system's internals work
38  * together, in order to facilitate future development and custom
39  * modifications to the system. The following entry points to the
40  * documentation might be useful.
41  *
42  * <dl>
43  * <dt>General definitions</dt>
44  * <dd><dl>
45  *   <dt>types.hpp file</dt>
46  *   <dd>Global types in DiCon.</dd>
47  *   <dt>log.hpp file</dt>
48  *   <dd>Global log system to report debug and error information.</dd>
49  *   <dt>specifier.hpp file</dt>
50  *   <dd>Argument and schedule specifiers in main configuration
51  *     file.</dd>
52  * </dl></dd>
53  * <dt>%Job handling</dt>
54  * <dd><dl>
55  *   <dt>Job structure</dt>
56  *   <dd>Main structure to hold each %optimizer job defined in any
57  *     DiCon run.</dd>
58  *   <dt>LazySet and LazyValue interfaces</dt>
59  *   <dd>Two lazy constructs used to create concrete job objects
60  *     on-the-fly.</dd>
61  * </dl></dd>
62  * <dt>Node managers</dt>
63  * <dd><dl>
64  *   <dt>NodeManager, MainNodeManager, and ProcNodeManager
65  *     interfaces</dt>
66  *   <dd>Split of processing cluster into main node and processing
67  *     nodes.</dd>
68  *   <dt>message.hpp file</dt>
69  *   <dd>Actions available at processing node for the main node to
70  *     call.</dd>
71  * </dl></dd>
72  * <dt>External interfaces</dt>
73  * <dd><dl>
74  *   <dt>Communicator class</dt>
75  *   <dd>Message-oriented protocol for simulators.</dd>
76  *   <dt>Simulator class</dt>
77  *   <dd>Interface that custom simulators must fulfill.</dd>
78  *   <dt>optimizer namespace</dt>
79  *   <dd>Interface that custom %optimizers must fulfill.</dd>
80  * </dl></dd>
81  * </dl>
82  */
83 /**
84  * @brief Main entry point.
85  *
86  * The main function is the main entry point into the
87  * program. Execution of everything starts here.
88  *
89  * @param argc Command-line argument count.
90  * @param argv Command-line argument array.
91  * @returns Process return value.
92  */
93 int main( int argc, char *argv[] );
94
95 #endif //DICON_MAIN_HPP_
```

Listing B.24: src/main.cpp

```
 1 /*--- [Distribution] ------------------------------------------------
 2  * This file is part of the Disease Control System DiCon.
 3  *
 4  * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
 5  * Designed and developed with the guidance of Nedialko B. Dimitrov
 6  * and Lauren Ancel Meyers at the University of Texas at Austin.
 7  *
 8  * DiCon is free software: you can redistribute it and/or modify it
 9  * under the terms of the GNU General Public License as published by
10  * the Free Software Foundation, either version 3 of the License, or
11  * (at your option) any later version.
12  *
13  * DiCon is distributed in the hope that it will be useful, but
14  * WITHOUT ANY WARRANTY; without even the implied warranty of
15  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
16  * General Public License for more details.
17  *
18  * You should have received a copy of the GNU General Public License
19  * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
20  *----------------------------------------------------------------*/
21
22 #include "main.hpp"
23 #include "error.hpp"
24 #include "file.hpp"
25 #include "log.hpp"
26 #include "manager.hpp"
27 #include "specifier.hpp"
28 #include "version.hpp"
29 #include <boost/format.hpp>
30 #include <boost/function.hpp>
```

```
#include <boost/mpi/communicator.hpp>
#include <boost/mpi/environment.hpp>
#include <boost/version.hpp>
#include <csignal>
#include <cerrno>


namespace detail {

  static
  const char
  usage_text[] =
    "Disease Control System DiCon (%2$s)"          "\n"
    ""                                             "\n"
    "Copyright (C) 2009  Sebastian Goll, University of Texas at Austin" "\n"
    "Designed and developed with the guidance of Nedialko B. Dimitrov" "\n"
    "and Lauren Ancel Meyers at the University of Texas at Austin." "\n"
    ""                                             "\n"
    "Usage: %1$s [job_directory]"                  "\n"
    "       %1$s -a [argument_specifier]"          "\n"
    "       %1$s -s [schedule_specifier]"          "\n"
    ""                                             "\n"
    "This program comes with ABSOLUTELY NO WARRANTY."
    ;


  static
  void
  deactivate_sigpipe() {
    struct sigaction act;
    memset( &act, 0, sizeof(act) );

    act.sa_handler = SIG_IGN;
    sigemptyset( &act.sa_mask );
    act.sa_flags = 0;

    if( sigaction(SIGPIPE, &act, NULL) != 0 )
      BOOST_THROW_EXCEPTION( SignalError() << errinfo_api_function("sigaction") << errinfo_errno(errno) );
  }


  template< typename T >
  static
  void
  list_specifier( const std::string &specifier, T (*parser)(const std::string &) ) {
    T set;

    try {
      set = parser( specifier );
    }
    catch( Error &e ) {
      std::cerr << (boost::format("Failed to parse specifier \"%s\".\n\n%s") % specifier % e.what() % boost::diagnostic_information(e));
      std::cerr << std::flush;
      return;
    }

    std::cout << (boost::format("Listing specifier \"%s\".\n") % specifier) << std::endl;

    size_t i = 0;
    for( set->reset(); set->has(); set->inc() ) {
      typename detail::LazySetPtr<T>::value_t value;

      try {
        ++i;
        value = set->get();
      }
      catch( Error &e ) {
        std::cout << std::flush;
        std::cerr << (boost::format("Failed to get element #%u.\n\n%s") % i % e.what() % boost::diagnostic_information(e));
        std::cerr << std::flush;
        return;
      }

      std::cout << '#' << i << ": " << value << '\n';
    }

    if( i != 0 )
      std::cout << '\n';

    std::cout << (boost::format("Specifier has %u elements.") % i) << std::endl;
  }

}


int
main( int argc, char *argv[] ) {
  // First do some general things without MPI.

  if( argc == 3 && std::string(argv[1]) == "-a" ) {
    detail::list_specifier( argv[2], &parse_argument_specifier );
    return 0;
  }
  else if( argc == 3 && std::string(argv[1]) == "-s" ) {
    detail::list_specifier( argv[2], &parse_schedule_specifier );
    return 0;
  }

  if( argc != 2 ) {
#ifdef DICON_PROGRAM_NAME
    const char program_name[] = DICON_PROGRAM_NAME;
#else
    const char *program_name = argv[0];
#endif
    std::cerr << (boost::format(detail::usage_text)
                  % program_name % program_version) << std::endl;
    return -1;
  }
```

```
142   // Here begins the MPI part of the program.
143
144   boost::mpi::environment environment( argc, argv );
145
146   try {
147     File::chdir( std::string(argv[1]) );
148     detail::deactivate_sigpipe();
149
150     boost::mpi::communicator world;
151     NodeManager::create(world)->run();
152   }
153   catch( NodeManagerShutdownError & ) {
154     // Don't show error message again.
155     environment.abort( -1 );
156     return -1;
157   }
158 #if BOOST_VERSION >= 103900
159   catch( ... ) {
160     log_failure( "Caught unhandled exception in main().\n" +
161                  boost::current_exception_diagnostic_information() );
162     environment.abort( -1 );
163     return -1;
164   }
165 #else
166   catch( boost::exception &e ) {
167     log_failure( "Caught unhandled exception in main().\n" +
168                  boost::diagnostic_information(e) );
169     environment.abort( -1 );
170     return -1;
171   }
172   catch( ... ) {
173     log_failure( "Caught unhandled exception in main().\n" +
174                  std::string("<unknown exception>\n") );
175     environment.abort( -1 );
176     return -1;
177   }
178 #endif
179
180   return 0;
181 }
```

Listing B.25: src/manager.hpp

```
1  #ifndef DICON_MANAGER_HPP_
2  #define DICON_MANAGER_HPP_
3
4  /*--- [Distribution] ---------------------------------------------------
5   * This file is part of the Disease Control System DiCon.
6   *
7   * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8   * Designed and developed with the guidance of Nedialko B. Dimitrov
9   * and Lauren Ancel Meyers at the University of Texas at Austin.
10  *
11  * DiCon is free software: you can redistribute it and/or modify it
12  * under the terms of the GNU General Public License as published by
13  * the Free Software Foundation, either version 3 of the License, or
14  * (at your option) any later version.
15  *
16  * DiCon is distributed in the hope that it will be useful, but
17  * WITHOUT ANY WARRANTY; without even the implied warranty of
18  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19  * General Public License for more details.
20  *
21  * You should have received a copy of the GNU General Public License
22  * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23  *-----------------------------------------------------------------------*/
24
25 /**
26  * @file
27  * @brief NodeManager interface.
28  */
29 #include "error.hpp"
30 #include "stats.hpp"
31 #include <boost/noncopyable.hpp>
32 #include <boost/shared_ptr.hpp>
33
34
35 /// Node manager related error.
36 struct NodeManagerError : virtual Error
37 { virtual const char *what() const throw() { return "Node manager ↩
    related error."; } };
38
39 /// Shutting down node manager failed.
40 struct NodeManagerShutdownError : virtual NodeManagerError
41 { virtual const char *what() const throw() { return "Shutting down ↩
    node manager failed."; } };
42
43
44 namespace boost {
45   namespace mpi {
46     class communicator;
47   }
48 }
49
50 /**
51  * @brief Node manager interface.
52  *
53  * The NodeManager class represents the node manager running on each
54  * MPI node in the computing cluster. It is the abstract base class to
55  * either the manager running on the main node (MainNodeManager
56  * interface, MainNodeManagerImpl class) or the ones running on the
57  * processing nodes (ProcNodeManager interface, ProcNodeManagerImpl
58  * class).
59  */
60 class NodeManager
61   : boost::noncopyable
62 {
63 public:
64   /**
65    * @brief Run node manager.
66    *
67    * Run the node manager.  This method implements the node manager's
68    * specific function (either main node manager or processing node
69    * manager) and can be though of as the main function of the node
70    * manager. It should only be called once during the lifetime of the
71    * NodeManager object.
72    */
73   virtual void run() = 0;
74
75   virtual ~NodeManager() {}
```

```
76
77    /**
78     * @brief Create node manager.
79     *
80     * Create a new node manager object. This static method creates
81     * either an object of the implementing MainNodeManagerImpl class or
82     * ProcNodeManagerImpl class, depending on the current node's rank
83     * within the MPI communicator context given by @e world: when the
84     * node with rank 0 calls this method, a main node manager will be
85     * created, otherwise a processing node manager will be created.
86     *
87     * @param world MPI communicator.
88     * @returns Concrete node manager.
89     */
90    static boost::shared_ptr<NodeManager>
91    create( boost::mpi::communicator &world );
92
93  protected:
94    /**
95     * @brief %Statistics helper.
96     *
97     * The StatsEntry class within the NodeManager class is used exactly
98     * like the global @link ::StatsEntry StatsEntry@endlink class,
99     * except that its first argument on construction is a NodeManager
100    * object. This NodeManager provides and encapsulates the actual
101    * Statistics object necessary for the @link ::StatsEntry
102    * StatsEntry@endlink object to work on. The statistics collected
103    * by using the StatsEntry class are output when calling the
104    * log_statistics() method.
105    */
106   class StatsEntry
107     : public ::StatsEntry
108   {
109   public:
110     /**
111      * @brief Push node onto tree.
112      *
113      * Constructor that pushes another node onto the statistics tree
114      * defined by the Statistics object encapsulated in the
115      * NodeManager object given as @e manager. Apart from the
116      * difference in the first argument, this constructor behaves like
117      * the one of the global @link ::StatsEntry StatsEntry@endlink
118      * class.
119      *
120      * @param manager NodeManager object.
121      * @param what Node name.
122      * @param priority Node priority.
123      * @param idle Idle node name.
124      * @param idle_priority Idle node priority.
125      *
126      * @throws TimeError when getting current time failed.
127      */
128     StatsEntry( NodeManager &manager, const std::string &what, int
    priority = 0,
129                 const std::string &idle = std::string(), int
    idle_priority = 0 );
130   };
131
132 protected:
133   /**
134    * @brief Create node manager.
135    *
136    * Constructor that creates a new node manager. This constructor is
137    * protected as the NodeManager class is an abstract base class that
138    * does not allow direct instantiation.
139    *
140    * @param world MPI communicator.
141    */
142   NodeManager( boost::mpi::communicator &world );
143
144   /**
145    * @brief Get MPI communicator.
146    *
147    * Get the MPI communicator used on construction.
148    *
149    * @returns MPI communicator.
150    */
151   const boost::mpi::communicator &world() const;
152   /**
153    * @brief Get MPI communicator.
154    *
155    * Get the MPI communicator used on construction.
156    *
157    * @returns MPI communicator.
158    */
159   boost::mpi::communicator &world();
160
161   /**
162    * @brief Print statistics.
163    *
164    * Print some statistics concerning this node manager to the global
165    * log system. The %message will be logged using the info log level
166    * (@link ::LOG_INFO LOG_INFO@endlink).
167    */
168   void log_statistics();
169
170   /**
171    * @brief Get minimum tag value.
172    *
173    * Get the minimum tag value that can be used in MPI communication.
174    *
175    * @returns Minimum tag value.
176    */
177   static int min_tag();
178   /**
179    * @brief Get maximum tag value.
180    *
181    * Get the maximum tag value that can be used in MPI communication.
182    *
183    * @returns Maximum tag value.
184    */
185   static int max_tag();
186   /**
187    * @brief Get number of tag values.
188    *
189    * Get the total number of different tag values available to be used
190    * in MPI communication. The actual values are the ones within the
191    * range defined by min_tag() and max_tag(). The range includes
192    * both endpoints.
193    *
194    * @returns Number of tag values.
195    */
```

```
196     static unsigned tag_count();
197     /**
198      * @brief Get next tag value.
199      *
200      * Get the next tag available to be used in MPI communication. The
201      * current tag value given by @e tag must be within the available
202      * range defined by min_tag() and max_tag(), otherwise this call
203      * fails. The range includes both endpoints.
204      *
205      * @param tag Current tag value.
206      * @returns Next available tag value.
207      */
208     static int &next_tag( int &tag );
209
210   private:
211     boost::mpi::communicator &world_;
212     Statistics stats_;
213   };
214
215
216   #include "manager.ipp"
217
218   #endif //DICON_MANAGER_HPP_
```

Listing B.26: src/manager.ipp

```
1   // -*- c++ -*-
2
3   /*---- [Distribution] ------------------------------------------------
4    * This file is part of the Disease Control System DiCon.
5    *
6    * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
7    * Designed and developed with the guidance of Nedialko B. Dimitrov
8    * and Lauren Ancel Meyers at the University of Texas at Austin.
9    *
10   * DiCon is free software: you can redistribute it and/or modify it
11   * under the terms of the GNU General Public License as published by
12   * the Free Software Foundation, either version 3 of the License, or
13   * (at your option) any later version.
14   *
15   * DiCon is distributed in the hope that it will be useful, but
16   * WITHOUT ANY WARRANTY; without even the implied warranty of
17   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
18   * General Public License for more details.
19   *
20   * You should have received a copy of the GNU General Public License
21   * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
22   *--------------------------------------------------------------------*/
23
24  inline
25  const boost::mpi::communicator &
26  NodeManager::world() const {
27    return world_;
28  }
29
30
31  inline
32  boost::mpi::communicator &
33  NodeManager::world() {
34    return world_;
35  }
```

Listing B.27: src/manager.cpp

```
1   /*---- [Distribution] ------------------------------------------------
2    * This file is part of the Disease Control System DiCon.
3    *
4    * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
5    * Designed and developed with the guidance of Nedialko B. Dimitrov
6    * and Lauren Ancel Meyers at the University of Texas at Austin.
7    *
8    * DiCon is free software: you can redistribute it and/or modify it
9    * under the terms of the GNU General Public License as published by
10   * the Free Software Foundation, either version 3 of the License, or
11   * (at your option) any later version.
12   *
13   * DiCon is distributed in the hope that it will be useful, but
14   * WITHOUT ANY WARRANTY; without even the implied warranty of
15   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
16   * General Public License for more details.
17   *
18   * You should have received a copy of the GNU General Public License
19   * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
20   *--------------------------------------------------------------------*/
21
22  #include "manager.hpp"
23  #include "error.hpp"
24  #include "manager_main_impl.hpp"
25  #include "manager_proc_impl.hpp"
26  #include <boost/mpi/communicator.hpp>
27  #include <boost/mpi/environment.hpp>
28
29
30  NodeManager::StatsEntry::StatsEntry( NodeManager &manager,
31                                       const std::string &what, int
    priority,
32                                       const std::string &idle, int
    idle_priority )
33    : ::StatsEntry(manager.stats_, what, priority, idle, idle_priority)
34  {
35  }
36
37
38  NodeManager::NodeManager( boost::mpi::communicator &world )
39    : world_(world)
40  {
41  }
42
43
44  boost::shared_ptr<NodeManager>
45  NodeManager::create( boost::mpi::communicator &world ) {
46    if( world.rank() == 0 )
47      return boost::shared_ptr<NodeManager>( new MainNodeManagerImpl(
    world) );
48    else
49      return boost::shared_ptr<NodeManager>( new ProcNodeManagerImpl(
    world) );
50  }
```

```
51
52
53  void
54  NodeManager::log_statistics() {
55    std::stringstream buffer;
56    stats_.print( buffer );
57    log_info( (boost::format("Time_statistics:\n%s") % buffer.str()).str()
      () );
58  }
59
60
61  int
62  NodeManager::min_tag() {
63    return 0;
64  }
65
66
67  int
68  NodeManager::max_tag() {
69    return boost::mpi::environment::max_tag();
70  }
71
72
73  unsigned
74  NodeManager::tag_count() {
75    assert( min_tag() <= max_tag() );
76    return unsigned(max_tag() - min_tag()) + 1;
77  }
78
79
80  int &
81  NodeManager::next_tag( int &tag ) {
82    DICON_ASSERT_RANGE( tag, min_tag(), max_tag() );
83
84    if( tag != max_tag() )
85      ++tag;
86    else
87      tag = min_tag();
88
89    return tag;
90  }
```

Listing B.28: src/manager_main.hpp

```
1  #ifndef DICON_MANAGER_MAIN_HPP_
2  #define DICON_MANAGER_MAIN_HPP_
3
4  /*---- [Distribution] ------------------------------------------------
5   * This file is part of the Disease Control System DiCon.
6   *
7   * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8   * Designed and developed with the guidance of Nedialko B. Dimitrov
9   * and Lauren Ancel Meyers at the University of Texas at Austin.
10  *
11  * DiCon is free software: you can redistribute it and/or modify it
12  * under the terms of the GNU General Public License as published by
13  * the Free Software Foundation, either version 3 of the License, or
14  * (at your option) any later version.
15  *
16  * DiCon is distributed in the hope that it will be useful, but
17  * WITHOUT ANY WARRANTY; without even the implied warranty of
18  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19  * General Public License for more details.
20  *
21  * You should have received a copy of the GNU General Public License
22  * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23  *--------------------------------------------------------------------*/
24
25 /**
26  * @file
27  * @brief MainNodeManager interface.
28  */
29 #include "manager.hpp"
30 #include "message.hpp"
31 #include "request.hpp"
32 #include <queue>
33
34
35 /**
36  * @brief Node state.
37  *
38  * The NodeState enum describes the state any given processing node is
39  * currently in.  It can be queried using the @link
40  * MainNodeManager::state() state()@endlink method of the
41  * MainNodeManager class.  Except for @link ::NODE_IDLE
42  * NODE_IDLE@endlink, the node states correspond directly to the
43  * messages used for node intercommunication, as defined in the
44  * documentation of the message.hpp file.
45  */
46 enum NodeState
47 { NODE_IDLE                 /// Node is idle.
48 , NODE_STARTING_LOGGING     /// Node is starting logging.
49 , NODE_INITIALIZING_OPTIMIZER /// Node is initializing %optimizer.
50 , NODE_INITIALIZING_SIMULATOR /// Node is initializing simulator.
51 , NODE_INITIALIZING_COMBINED  /// Node is initializing combined %
   optimizer/simulator.
52 , NODE_GETTING_POLICY       /// Node is getting policy (regular %
   optimizer only).
53 , NODE_SIMULATING           /// Node is simulating (simulator only).
54 , NODE_UPDATING            /// Node is updating (regular %optimizer
   only).
55 , NODE_STEPPING_COMBINED    /// Node is stepping (combined %
   optimizer/simulator only).
56 , NODE_DUMPING_OPTIMIZER    /// Node is dumping %optimizer state (%
   optimizer only).
57 , NODE_DUMPING_POLICIES     /// Node is dumping policies (%optimizer
   only).
58 , NODE_SHUTTING_DOWN        /// Node is shutting down.
59 };
60
61 /// Output textual representation of node state.
62 std::ostream &operator<<( std::ostream &out, NodeState state );
63
64
65 namespace detail {
66   class AnswerDispatcher;
67 }
68
69 /**
70  * @brief Main node manager interface.
```

```
 *
 * The MainNodeManager class provides the interface to the node
 * manager running on the main node in the computing cluster. As such,
 * it provides methods for sending messages to the processing nodes in
 * the cluster, and must be filled in with implementations of how the
 * main node manager reacts when each such request has finished.
 *
 * Each of those methods and each of the virtual methods to be
 * implemented corresponds to the %message with the same name used for
 * node intercommunication, as defined in the documentation of the
 * message.hpp class, and the matching structures defined in the
 * message::question and message::answer namespaces. See the
 * documentation that is given there for details on each available
 * %message and its parameters.
 *
 * Calling one of the methods start_logging(), init_optimizer(),
 * init_simulator(), init_combined(), get_policy(), simulate(),
 * update(), step_combined(), dump_optimizer(), dump_policies(),
 * shutdown() sends the corresponding %message to the given processing
 * node, returning immediately to the caller, i.e., the request is
 * sent in a non-blocking way. Calling process() waits for any one of
 * the pending requests to finish. When this happens, the
 * corresponding <code>finish_</code> method is called. If the
 * request failed, e.g., because an uncaught exception was thrown, the
 * handle_failure() method is called instead; in this case, state()
 * can be used to figure out which request failed.
 *
 * More than one request can be sent to a processing node
 * simultaneously. This way, a primitive backlog can be realized; the
 * processing node is guaranteed to receive the requests in the order
 * they are sent. After a number of requests have been sent, the
 * corresponding <code>finish_</code> methods are called in the same
 * order, with handle_failure() taking the place of any
 * <code>finish_</code> method whenever a request failed.
 *
 * A request can be sent to a processing node at any time. In
 * particular, new requests can be initiated during the execution of
 * any of the <code>finish_</code> methods and handle_failure().
 */
class MainNodeManager
  : public NodeManager
{
  friend class detail::AnswerDispatcher;

protected:
  /**
   * @brief Create main node manager.
   *
   * Constructor that creates a new node manager. This constructor is
   * protected as the MainNodeManager class is an abstract base class
   * that does not allow direct instantiation.
   *
   * @param world MPI communicator.
   */
  MainNodeManager( boost::mpi::communicator &world );

public:
  virtual ~MainNodeManager();

protected:
  /**
   * @brief Get minimum child ID.
   *
   * Get the node ID within the MPI communicator given on construction
   * that corresponds to the first processing node. This is guaranteed
   * to be 1.
   *
   * The behavior of this method is not defined when child_count()
   * returns 0.
   *
   * @returns Minimum child ID.
   */
  int min_child() const;
  /**
   * @brief Get maximum child ID.
   *
   * Get the node ID within the MPI communicator given on construction
   * that corresponds to the last processing node.
   *
   * The behavior of this method is not defined when child_count()
   * returns 0.
   *
   * @returns Maximum child ID.
   */
  int max_child() const;

  /**
   * @brief Get number of children.
   *
   * Get the number of processing nodes within the MPI communicator
   * given on construction. When this is 0, only the main node
   * exists. In this case, the behavior of calling min_child() and
   * max_child() is not defined.
   *
   * @returns Number of children.
   */
  unsigned child_count() const;

protected:
  /// Start logging on processing node given by @e node.
  void start_logging ( int node, const boost::filesystem::path &logfile, LogLevel min_level );
  /// Initialize %optimizer on processing node given by @e node.
  void init_optimizer( int node
                     , const boost::filesystem::path &optimizer_logfile
                     , const boost::filesystem::path &simulator_logfile
                     , const std::string &optimizer_library, const arguments_t &optimizer_arguments
                     , const std::string &simulator_command, const arguments_t &simulator_arguments
                     , const boost::optional<boost::filesystem::path> &optimizer_map_file
                     , const boost::optional<boost::filesystem::path> &optimizer_lib_file );
  /// Initialize simulator on processing node given by @e node.
  void init_simulator( int node
                     , const boost::filesystem::path &simulator_logfile
                     , const std::string &simulator_command, const arguments_t &simulator_arguments );
```

```
184   /// Initialize combined %optimizer/simulator on processing node
185   /// given by @e node.
186   void init_combined ( int node
187                      , const boost::filesystem::path &
      optimizer_logfile
188                      , const boost::filesystem::path &
      simulator_logfile
189                      , const std::string &optimizer_library, const
      arguments_t &optimizer_arguments
190                      , const std::string &simulator_command, const
      arguments_t &simulator_arguments
191                      , const boost::optional<boost::filesystem::path>
      &optimizer_map_file
192                      , const boost::optional<boost::filesystem::path>
      &optimizer_lib_file           );
193   /// Get next policy on processing node given by @e node.
194   void get_policy     ( int node
                                    );
195   /// Simulate policy on processing node given by @e node.
196   void simulate       ( int node, const policy_t &policy
                                    );
197   /// Update reward on processing node given by @e node.
198   void update         ( int node, const policy_t &policy, double reward
                                    );
199   /// Take step on processing node given by @e node.
200   void step_combined ( int node
                                    );
201   /// Dump %optimizer state on processing node given by @e node.
202   void dump_optimizer( int node, const boost::filesystem::path &file,
      DumpOptimizerMode mode        );
203   /// Dump list of best policies on processing node given by @e node.
204   void dump_policies ( int node, const boost::filesystem::path &file,
      unsigned count, bool display  );
205   /// Shutdown on processing node given by @e node.
206   void shutdown       ( int node
                                    );
207
208 protected:
209   /**
210    * @brief Get node state.
211    *
212    * Get the current state of the node given by @e node. If this
213    * method is called during <code>handle_failure</code> or one of the
214    * <code>finish_</code> methods, the return value describes the
215    * request that is currently being processed. Otherwise, it is
216    * equivalent to the first request this node will return from, as
217    * defined in the description of the MainNodeManager class, or @link
218    * ::NODE_IDLE NODE_IDLE@endlink when this node has no pending
219    * requests.
220    *
221    * @param node Node.
222    * @returns Node state.
223    *
224    * @throws AssertionError when @e node is not within the range
225    *   defined by min_child() and max_child().
226    */
227   NodeState state( int node ) const;
228
229 protected:
230   /// Finish starting logging on processing node given by @e node.
231   virtual void finish_start_logging ( int node
                                     ) = 0;
232   /// Finish initializing %optimizer on processing node given by @e
233   /// node.
234   virtual void finish_init_optimizer( int node
                                     ) = 0;
235   /// Finish initializing simulator on processing node given by @e
236   /// node.
237   virtual void finish_init_simulator( int node
                                     ) = 0;
238   /// Finish initializing combined %optimizer/simulator on processing
239   /// node given by @e node.
240   virtual void finish_init_combined ( int node
                                     ) = 0;
241   /// Finish getting next policy on processing node given by @e node.
242   virtual void finish_get_policy    ( int node, const boost::optional<
      policy_t> &policy ) = 0;
243   /// Finish simulating policy on processing node given by @e node.
244   virtual void finish_simulate      ( int node, double reward
                                     ) = 0;
245   /// Finish updating reward on processing node given by @e node.
246   virtual void finish_update        ( int node
                                     ) = 0;
247   /// Finish taking step on processing node given by @e node.
248   virtual void finish_step_combined ( int node, bool got_policy
                                     ) = 0;
249   /// Finish dumping %optimizer state on processing node given by @e
250   /// node.
251   virtual void finish_dump_optimizer( int node
                                     ) = 0;
252   /// Finish dumping policies on processing node given by @e node.
253   virtual void finish_dump_policies ( int node
                                     ) = 0;
254   /// Finish shutting down on processing node given by @e node.
255   virtual void finish_shutdown      ( int node, bool done
                                     ) = 0;
256
257   /**
258    * @brief Handle failure while processing request.
259    *
260    * Handle the failure that occured while processing the request on
261    * the processing node given by @e node. This method is called in
262    * place of the corresponding <code>finish_</code> method whenever a
263    * request failed to complete on a processing node, as defined in
264    * the description of the MainNodeManager class. The parameter @e
265    * what contains a detailed description of the error (though not
266    * necessarily in a user friendly format).
267    *
268    * @param node Node that failed.
269    * @param what Description of error.
270    */
271   virtual void handle_failure       ( int node, const std::string &
      what                          ) = 0;
272
273   /**
274    * @brief Wait for and finish one request.
275    *
276    * Wait for any one of the pending requests sent to a processing
277    * node and finish it. This call blocks until at least one of the
278    * requests asynchronously sent to any processing node returns. It
279    * then calls the corresponding <code>finish_</code> method or
```

```
280     * handle_failure(). Requests for any single node are guaranteed to
281     * both be processed by the processing node and finished by this
282     * method in the exact order they were sent. Requests to different
283     * nodes are allowed to be finished in any order.
284     *
285     * @returns @c false iff no request is pending.
286     */
287   bool process();
288
289 private:
290   typedef std::queue<NodeState> state_queue_t;
291
292   int &current_tag( int node );
293   state_queue_t &state_queue( int node );
294   const state_queue_t &state_queue( int node ) const;
295
296   template< typename Q >
297   void async_enqueue( int node, const Q &question, NodeState state );
298
299 private:
300   struct RequestData {
301     boost::shared_ptr<message::Answer> answer;
302     int which_message;
303   };
304
305   struct NodeRecord {
306     int current_tag;
307     state_queue_t state_queue;
308
309     NodeRecord( int min_tag );
310   };
311
312 private:
313   std::deque<boost::mpi::request> isend_queue_;
314   RequestQueue<RequestData> request_queue_;
315   std::vector<NodeRecord> nodes_;
316 };
317
318 #endif //DICON_MANAGER_MAIN_HPP_
```

Listing B.29: src/manager_main.cpp

```
1  /*---- [Distribution] ------------------------------------------------
2   * This file is part of the Disease Control System DiCon.
3   *
4   * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
5   * Designed and developed with the guidance of Nedialko B. Dimitrov
6   * and Lauren Ancel Meyers at the University of Texas at Austin.
7   *
8   * DiCon is free software: you can redistribute it and/or modify it
9   * under the terms of the GNU General Public License as published by
10  * the Free Software Foundation, either version 3 of the License, or
11  * (at your option) any later version.
12  *
13  * DiCon is distributed in the hope that it will be useful, but
14  * WITHOUT ANY WARRANTY; without even the implied warranty of
15  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
16  * General Public License for more details.
17  *
18  * You should have received a copy of the GNU General Public License
19  * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
20  *--------------------------------------------------------------------*/
21
22 #include "manager_main.hpp"
23 #include "error.hpp"
24 #include "log.hpp"
25 #include <boost/format.hpp>
26 #include <boost/mpi/communicator.hpp>
27
28
29 namespace detail {
30
31   static const size_t isend_queue_size = 50;
32
33 }
34
35
36
37 namespace detail {
38
39   class AnswerDispatcher
40     : public boost::static_visitor<>
41   {
42   public:
43     AnswerDispatcher( MainNodeManager &manager, const boost::mpi::
   status &status );
44
45   public:
46     void operator()( const message::answer::StartLogging  &data )
   const;
47     void operator()( const message::answer::InitOptimizer &data )
   const;
48     void operator()( const message::answer::InitSimulator &data )
   const;
49     void operator()( const message::answer::InitCombined  &data )
   const;
50     void operator()( const message::answer::GetPolicy     &data )
   const;
51     void operator()( const message::answer::Simulate      &data )
   const;
52     void operator()( const message::answer::Update        &data )
   const;
53     void operator()( const message::answer::StepCombined  &data )
   const;
54     void operator()( const message::answer::DumpOptimizer &data )
   const;
55     void operator()( const message::answer::DumpPolicies  &data )
   const;
56     void operator()( const message::answer::Shutdown      &data )
   const;
57
58     void operator()( const message::answer::Failure       &data )
   const;
59
60   private:
61     MainNodeManager &manager_;
62     const boost::mpi::status &status_;
63   };
64
65 }
```

```
66
67
68
69  namespace detail {
70
71    template< typename Q >
72    static
73    message::Question
74    make_question( const Q &question ) {
75      message::Question res;
76      res.data = question;
77      return res;
78    }
79
80
81    static
82    inline
83    void
84    log_debug( const std::string &what, int node, bool finish = false ) throw() {
85      try {
86        ::log_debug( (boost::format( !finish
87                                     ? "[Node_#%u]_%s()"
88                                     : "[Node_#%u]_finish_%s()" ) % node % what).str() );
89      }
90      catch( ... ) {
91        // Ignore.
92      }
93    }
94
95
96    AnswerDispatcher::AnswerDispatcher( MainNodeManager &manager, const boost::mpi::status &status )
97      : manager_(manager), status_(status)
98    {
99    }
100
101
102   void
103   AnswerDispatcher::operator()( const message::answer::StartLogging &data ) const {
104     MainNodeManager::StatsEntry se( manager_, "start_logging()", 10 );
105     log_debug( "start_logging", status_.source(), true );
106     manager_.finish_start_logging( status_.source() );
107   }
108
109
110   void
111   AnswerDispatcher::operator()( const message::answer::InitOptimizer &data ) const {
112     MainNodeManager::StatsEntry se( manager_, "init_optimizer()", 20 );
113     log_debug( "init_optimizer", status_.source(), true );
114     manager_.finish_init_optimizer( status_.source() );
115   }
116
117
118   void
119   AnswerDispatcher::operator()( const message::answer::InitSimulator &data ) const {
120     MainNodeManager::StatsEntry se( manager_, "init_simulator()", 30 );
121     log_debug( "init_simulator", status_.source(), true );
122     manager_.finish_init_simulator( status_.source() );
123   }
124
125
126   void
127   AnswerDispatcher::operator()( const message::answer::InitCombined &data ) const {
128     MainNodeManager::StatsEntry se( manager_, "init_combined()", 40 );
129     log_debug( "init_combined", status_.source(), true );
130     manager_.finish_init_combined( status_.source() );
131   }
132
133
134   void
135   AnswerDispatcher::operator()( const message::answer::GetPolicy &data ) const {
136     MainNodeManager::StatsEntry se( manager_, "get_policy()", 50 );
137     log_debug( "get_policy", status_.source(), true );
138     manager_.finish_get_policy( status_.source(), data.policy );
139   }
140
141
142   void
143   AnswerDispatcher::operator()( const message::answer::Simulate &data ) const {
144     MainNodeManager::StatsEntry se( manager_, "simulate()", 60 );
145     log_debug( "simulate", status_.source(), true );
146     manager_.finish_simulate( status_.source(), data.reward );
147   }
148
149
150   void
151   AnswerDispatcher::operator()( const message::answer::Update &data ) const {
152     MainNodeManager::StatsEntry se( manager_, "update()", 70 );
153     log_debug( "update", status_.source(), true );
154     manager_.finish_update( status_.source() );
155   }
156
157
158   void
159   AnswerDispatcher::operator()( const message::answer::StepCombined &data ) const {
160     MainNodeManager::StatsEntry se( manager_, "step_combined()", 80 );
161     log_debug( "step_combined", status_.source(), true );
162     manager_.finish_step_combined( status_.source(), data.got_policy );
163   }
164
165
166   void
167   AnswerDispatcher::operator()( const message::answer::DumpOptimizer &data ) const {
168     MainNodeManager::StatsEntry se( manager_, "dump_optimizer()", 90 );
169     log_debug( "dump_optimizer", status_.source(), true );
170     manager_.finish_dump_optimizer( status_.source() );
171   }
```

```
172
173
174   void
175   AnswerDispatcher::operator()( const message::answer::DumpPolicies &data ) const {
176     MainNodeManager::StatsEntry se( manager_, "dump_policies()", 100 );
177     log_debug( "dump_policies", status_.source(), true );
178     manager_.finish_dump_policies( status_.source() );
179   }
180
181
182   void
183   AnswerDispatcher::operator()( const message::answer::Shutdown &data ) const {
184     MainNodeManager::StatsEntry se( manager_, "shutdown()", 110 );
185     log_debug( "shutdown", status_.source(), true );
186     manager_.finish_shutdown( status_.source(), data.done );
187   }
188
189
190   void
191   AnswerDispatcher::operator()( const message::answer::Failure &data ) const {
192     MainNodeManager::StatsEntry se( manager_, "handle_failure()", 900 );
193     log_debug( "handle_failure", status_.source() );
194     manager_.handle_failure( status_.source(), data.what );
195   }
196
197 }
198
199
200
201 std::ostream &
202 operator<<( std::ostream &out, NodeState state ) {
203   switch( state ) {
204 #define DICON_HANDLE_STATE_( STATE ) case STATE: out << #STATE; break
205   DICON_HANDLE_STATE_( NODE_IDLE                       );
206   DICON_HANDLE_STATE_( NODE_STARTING_LOGGING           );
207   DICON_HANDLE_STATE_( NODE_INITIALIZING_OPTIMIZER     );
208   DICON_HANDLE_STATE_( NODE_INITIALIZING_SIMULATOR     );
209   DICON_HANDLE_STATE_( NODE_INITIALIZING_COMBINED      );
210   DICON_HANDLE_STATE_( NODE_GETTING_POLICY             );
211   DICON_HANDLE_STATE_( NODE_SIMULATING                 );
212   DICON_HANDLE_STATE_( NODE_UPDATING                   );
213   DICON_HANDLE_STATE_( NODE_STEPPING_COMBINED          );
214   DICON_HANDLE_STATE_( NODE_DUMPING_OPTIMIZER          );
215   DICON_HANDLE_STATE_( NODE_DUMPING_POLICIES           );
216   DICON_HANDLE_STATE_( NODE_SHUTTING_DOWN              );
217 #undef DICON_HANDLE_STATE_
218   }
219
220   return out;
221 }
222
223
224 MainNodeManager::NodeRecord::NodeRecord( int min_tag )
225   : current_tag(min_tag)
226 {
227 }
228
229
230 MainNodeManager::MainNodeManager( boost::mpi::communicator &world )
231   : NodeManager(world), nodes_(child_count(), NodeRecord(min_tag()))
232 {
233 }
234
235
236 MainNodeManager::~MainNodeManager() {
237   try {
238     while( !isend_queue_.empty() ) {
239       try {
240         isend_queue_.front().wait();
241       }
242       catch( ... ) {
243         // Ignore.
244       }
245
246       isend_queue_.pop_front();
247     }
248   }
249   catch( ... ) {
250     // Ignore.
251   }
252 }
253
254
255 int
256 MainNodeManager::min_child() const {
257   return 1;
258 }
259
260
261 int
262 MainNodeManager::max_child() const {
263   return world().size() - 1;
264 }
265
266
267 unsigned
268 MainNodeManager::child_count() const {
269   return max_child() - min_child() + 1;
270 }
271
272
273 int &
274 MainNodeManager::current_tag( int node ) {
275   assert( min_child() <= node && node <= max_child() );
276   return nodes_[node - min_child()].current_tag;
277 }
278
279
280 MainNodeManager::state_queue_t &
281 MainNodeManager::state_queue( int node ) {
282   assert( min_child() <= node && node <= max_child() );
283   return nodes_[node - min_child()].state_queue;
284 }
285
286
287 const MainNodeManager::state_queue_t &
288 MainNodeManager::state_queue( int node ) const {
```

```
289   assert( min_child() <= node && node <= max_child() );
290   return nodes_[node - min_child()].state_queue;
291 }
292
293
294 template< typename Q >
295 void
296 MainNodeManager::async_enqueue( int node, const Q &question, NodeState
      state ) {
297   DICON_ASSERT_RANGE( node, min_child(), max_child() );
298
299   unsigned requests = request_queue_.size();
300   assert( requests <= tag_count() );
301   if( requests == tag_count() )
302     BOOST_THROW_EXCEPTION( AssertionError() );
303
304   assert( isend_queue_.size() <= detail::isend_queue_size );
305   if( isend_queue_.size() == detail::isend_queue_size ) {
306     assert( !isend_queue_.empty() );
307     isend_queue_.front().wait();
308     isend_queue_.pop_front();
309   }
310
311   const message::Question &q = detail::make_question( question );
312   const unsigned tag = next_tag(current_tag(node));
313
314   isend_queue_.push_back( world().isend(node, tag, q) );
315
316   RequestData data;
317   data.which_message = q.data.which();
318   data.answer.reset( new message::Answer() );
319
320   request_queue_.push( node,
321                        world().irecv(node, tag, *data.answer), data );
322
323   assert( state != NODE_IDLE );
324   state_queue(node).push( state );
325 }
326
327
328 void
329 MainNodeManager::start_logging( int node, const boost::filesystem::
      path &logfile, LogLevel min_level ) {
330   detail::log_debug( "start_logging", node );
331
332   message::question::StartLogging question;
333
334   question.logfile   = logfile;
335   question.min_level = min_level;
336
337   async_enqueue( node, question, NODE_STARTING_LOGGING );
338 }
339
340
341 void
342 MainNodeManager::init_optimizer( int node
343                                  , const boost::filesystem::path &
      optimizer_logfile
344                                  , const boost::filesystem::path &
      simulator_logfile
345                                  , const std::string &optimizer_library,
      const arguments_t &optimizer_arguments
346                                  , const std::string &simulator_command,
      const arguments_t &simulator_arguments
347                                  , const boost::optional<boost::
      filesystem::path> &optimizer_map_file
348                                  , const boost::optional<boost::
      filesystem::path> &optimizer_lib_file
349                                  )
350 {
351   detail::log_debug( "init_optimizer", node );
352
353   message::question::InitOptimizer question;
354
355   question.optimizer_logfile   = optimizer_logfile;
356   question.optimizer_library   = optimizer_library;
357   question.optimizer_arguments = optimizer_arguments;
358   question.optimizer_map_file  = optimizer_map_file;
359   question.optimizer_lib_file  = optimizer_lib_file;
360
361   question.simulator_logfile   = simulator_logfile;
362   question.simulator_command   = simulator_command;
363   question.simulator_arguments = simulator_arguments;
364
365   async_enqueue( node, question, NODE_INITIALIZING_OPTIMIZER );
366 }
367
368
369 void
370 MainNodeManager::init_simulator( int node
371                                  , const boost::filesystem::path &
      simulator_logfile
372                                  , const std::string &simulator_command,
      const arguments_t &simulator_arguments
373                                  )
374 {
375   detail::log_debug( "init_simulator", node );
376
377   message::question::InitSimulator question;
378
379   question.simulator_logfile   = simulator_logfile;
380   question.simulator_command   = simulator_command;
381   question.simulator_arguments = simulator_arguments;
382
383   async_enqueue( node, question, NODE_INITIALIZING_SIMULATOR );
384 }
385
386
387 void
388 MainNodeManager::init_combined( int node
389                                 , const boost::filesystem::path &
      optimizer_logfile
390                                 , const boost::filesystem::path &
      simulator_logfile
391                                 , const std::string &optimizer_library,
      const arguments_t &optimizer_arguments
392                                 , const std::string &simulator_command,
      const arguments_t &simulator_arguments
393                                 , const boost::optional<boost::
      filesystem::path> &optimizer_map_file
```

```
394                                , const boost::optional<boost::filesystem::path> &optimizer_lib_file
395                                )
396 {
397   detail::log_debug( "init_combined", node );
398
399   message::question::InitCombined question;
400
401   question.optimizer_logfile   = optimizer_logfile;
402   question.optimizer_library   = optimizer_library;
403   question.optimizer_arguments = optimizer_arguments;
404   question.optimizer_map_file  = optimizer_map_file;
405   question.optimizer_lib_file  = optimizer_lib_file;
406
407   question.simulator_logfile   = simulator_logfile;
408   question.simulator_command   = simulator_command;
409   question.simulator_arguments = simulator_arguments;
410
411   async_enqueue( node, question, NODE_INITIALIZING_COMBINED );
412 }
413
414
415 void
416 MainNodeManager::get_policy( int node ) {
417   detail::log_debug( "get_policy", node );
418
419   message::question::GetPolicy question;
420
421   // Nothing to do.
422
423   async_enqueue( node, question, NODE_GETTING_POLICY );
424 }
425
426
427 void
428 MainNodeManager::simulate( int node, const policy_t &policy ) {
429   detail::log_debug( "simulate", node );
430
431   message::question::Simulate question;
432
433   question.policy = policy;
434
435   async_enqueue( node, question, NODE_SIMULATING );
436 }
437
438
439 void
440 MainNodeManager::update( int node, const policy_t &policy, double reward ) {
441   detail::log_debug( "update", node );
442
443   message::question::Update question;
444
445   question.policy = policy;
446   question.reward = reward;
447
448   async_enqueue( node, question, NODE_UPDATING );
449 }
450
451
452 void
453 MainNodeManager::step_combined( int node ) {
454   detail::log_debug( "step_combined", node );
455
456   message::question::StepCombined question;
457
458   // Nothing to do.
459
460   async_enqueue( node, question, NODE_STEPPING_COMBINED );
461 }
462
463
464 void
465 MainNodeManager::dump_optimizer( int node, const boost::filesystem::path &file, DumpOptimizerMode mode ) {
466   detail::log_debug( "dump_optimizer", node );
467
468   message::question::DumpOptimizer question;
469
470   question.file = file;
471   question.mode = mode;
472
473   async_enqueue( node, question, NODE_DUMPING_OPTIMIZER );
474 }
475
476
477 void
478 MainNodeManager::dump_policies( int node, const boost::filesystem::path &file, unsigned count, bool display ) {
479   detail::log_debug( "dump_policies", node );
480
481   message::question::DumpPolicies question;
482
483   question.file = file;
484   question.count = count;
485   question.display = display;
486
487   async_enqueue( node, question, NODE_DUMPING_POLICIES );
488 }
489
490
491 void
492 MainNodeManager::shutdown( int node ) {
493   detail::log_debug( "shutdown", node );
494
495   message::question::Shutdown question;
496
497   // Nothing to do.
498
499   async_enqueue( node, question, NODE_SHUTTING_DOWN );
500 }
501
502
503 NodeState
504 MainNodeManager::state( int node ) const {
505   DICON_ASSERT_RANGE( node, min_child(), max_child() );
506
507   const state_queue_t &queue
508     = state_queue( node );
509
510   if( queue.empty() )
511     return NODE_IDLE;
```

```
512     else
513       return queue.front();
514   }
515
516
517   bool
518   MainNodeManager::process() {
519     if( request_queue_.empty() )
520       return false;
521
522     StatsEntry se( *this, "processing", 0, "(idle)", 0 );
523
524     const std::pair<boost::mpi::status, RequestData> &res
525       = request_queue_.wait();
526
527     const RequestData &data = res.second;
528     const message::Answer &answer = *data.answer;
529
530     if( answer.data.which() != data.which_message &&
531         !boost::get<message::answer::Failure>(&answer.data) )
532     {
533       // Answer type did not match question (and request did not fail).
534       BOOST_THROW_EXCEPTION( AssertionError()
535                              << errinfo_value<int>(answer.data.which())
536                              << errinfo_expected_value<int>(data.
    which_message) );
537     }
538
539     // Dispatch according to type of answer; call corresponding 'finish_
    ' method.
540     boost::apply_visitor( detail::AnswerDispatcher(*this, res.first),
    answer.data );
541
542     state_queue_t &queue
543       = state_queue( res.first.source() );
544
545     if( queue.empty() )
546       BOOST_THROW_EXCEPTION( AssertionError() );
547
548     queue.pop();
549
550     return true;
551   }
```

Listing B.30: src/manager_main_impl.hpp

```
1   #ifndef DICON_MANAGER_MAIN_IMPL_HPP_
2   #define DICON_MANAGER_MAIN_IMPL_HPP_
3
4   /*--- [Distribution] ---------------------------------------------------
5    * This file is part of the Disease Control System DiCon.
6    *
7    * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8    * Designed and developed with the guidance of Nedialko B. Dimitrov
9    * and Lauren Ancel Meyers at the University of Texas at Austin.
10   *
11   * DiCon is free software: you can redistribute it and/or modify it
12   * under the terms of the GNU General Public License as published by
13   * the Free Software Foundation, either version 3 of the License, or
14   * (at your option) any later version.
15   *
16   * DiCon is distributed in the hope that it will be useful, but
17   * WITHOUT ANY WARRANTY; without even the implied warranty of
18   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19   * General Public License for more details.
20   *
21   * You should have received a copy of the GNU General Public License
22   * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23   *--------------------------------------------------------------------*/
24
25  /**
26   * @file
27   * @brief MainNodeManagerImpl class.
28   */
29  #include "config.hpp"
30  #include "error.hpp"
31  #include "job_map.hpp"
32  #include "job_queue.hpp"
33  #include "log.hpp"
34  #include "manager_main.hpp"
35  #include <boost/shared_ptr.hpp>
36  #include <boost/tuple/tuple.hpp>
37
38
39  /// Checkpoint related error.
40  struct CheckpointError : virtual Error
41  { virtual const char *what() const throw() { return "Checkpoint_
    related_error."; } };
42
43  /// Checkpoint file does not match job.
44  struct CheckpointNoMatchError : virtual CheckpointError
45  { virtual const char *what() const throw() { return "Checkpoint_file_
    does_not_match_job."; } };
46
47
48  /**
49   * @brief Main node manager implementation.
50   *
51   * The MainNodeManagerImpl class implements the node manager running
52   * on the main node in the computing cluster. Specifically, it reads
53   * in and parses the main configuration file (using the Configuration
54   * and JobQueue classes), and processes each job defined therein by
55   * repeatedly spawning optimizers, simulators, and combined
56   * %optimizer/simulators on the processing nodes in the cluster as
57   * required, and managing the communication among these nodes, i.e.,
58   * it passes along policies from the optimizers to the simulators,
59   * collects resulting reward values, dumps %optimizer states to
60   * checkpoint files, and writes out the final result from each
61   * individual job (using the methods defined in the MainNodeManager
62   * interface).
63   *
64   * As defined in the NodeManager interface, the main node manager is
65   * run by calling the run() method.
66   */
67  class MainNodeManagerImpl
68    : public MainNodeManager
69  {
70    friend class NodeManager;
71
72  protected:
```

```
73    /**
74     * @brief Create main node manager.
75     *
76     * Constructor that creates a  new node manager. This constructor is
77     * protected since objects of the MainNodeManagerImpl class are only
78     * to be  costructed through the  static @link NodeManager::create()
79     * create()@endlink method of the NodeManager class.
80     *
81     * @param world MPI communicator.
82     */
83    MainNodeManagerImpl( boost::mpi::communicator &world );
84
85  protected:
86    virtual void finish_start_logging ( int node                          );
87    virtual void finish_init_optimizer( int node                          );
88    virtual void finish_init_simulator( int node                          );
89    virtual void finish_init_combined ( int node                          );
90    virtual void finish_get_policy    ( int node, const boost::optional<policy_t> &policy );
91    virtual void finish_simulate      ( int node, double reward           );
92    virtual void finish_update        ( int node                          );
93    virtual void finish_step_combined ( int node, bool got_policy         );
94    virtual void finish_dump_optimizer( int node                          );
95    virtual void finish_dump_policies ( int node                          );
96    virtual void finish_shutdown      ( int node, bool done               );
97
98    virtual void handle_failure       ( int node, const std::string &what );
99
100 public:
101   /**
102    * @brief Run main node manager.
103    *
104    * Run  the  main  node  manager.  This  method  handles  all  the
105    * communication  between  the  processing  nodes  in  the  computing
106    * cluster  as  necessary  in  order to  achieve  the  functionality
107    * outlined  in  the  description  of  the MainNodeManagerImpl class.
108    *
109    * This method should only be called once during the lifetime of the
110    * MainNodeManagerImpl  object.
111    *
112    * @throws NodeManagerShutdownError when  shutting down a processing
113    *   node gracefully failed.
114    */
115   virtual void run();
116
117 private:
118   void handle_failure_without_job   ( int node, const std::string &what );
119   void handle_failure_with_job      ( int node, const std::string &what );
120
121 private:
122   boost::filesystem::path job_logfile( const Job::ptr_t &job );
123   boost::filesystem::path checkpoint_file( const Job::ptr_t &job );
124   boost::filesystem::path opt_logfile( const Job::ptr_t &job, int node );
125   boost::filesystem::path sim_logfile( const Job::ptr_t &job, int node );
126   boost::filesystem::path optimizer_map_file( const Job::ptr_t &job, simcount_t simulation );
127   boost::filesystem::path optimizer_lib_file( const Job::ptr_t &job, simcount_t simulation );
128   boost::filesystem::path policy_bin_dumpfile( const Job::ptr_t &job, simcount_t simulation );
129   boost::filesystem::path policy_txt_dumpfile( const Job::ptr_t &job, simcount_t simulation );
130   boost::filesystem::path policy_bin_result( const Job::ptr_t &job );
131   boost::filesystem::path policy_txt_result( const Job::ptr_t &job );
132
133 private:
134   enum JobState
135   { JOB_INIT_OPTIMIZER
136   , JOB_RUNNING_NORMAL
137   , JOB_DUMPING_STATES
138   , JOB_DUMPING_RESULT
139   };
140
141   struct JobInfo {
142     JobInfo();
143
144     JobState state;
145     bool combined_node;
146
147     bool optimizer_initd;
148     bool ign_checkpoints;
149     bool getting_policy;                // [R]
150     bool out_of_policy;
151     bool starting_up;
152     bool failure;
153
154     unsigned dumping_count;
155     simcount_t total_policies;
156     simcount_t checkpoint_skips;
157     unsigned pending_step_calls;        // [C]
158
159     time_t checkpoint_time;
160     simcount_t checkpoint_sims;
161
162     std::string current_map_file;
163     std::string current_lib_file;
164
165     std::queue<policy_t> backlog;       // [R]
166     std::queue<policy_t> update_queue;  // [R]
167
168     boost::shared_ptr<LogFile> job_logfile;
169   };
170
171   struct NodeInfo {
172     NodeInfo();
173
174     bool simulator_initd;
```

```
175
176       std::queue<policy_t> simulate_queue;
177     };
178
179     struct Checkpoint {
180       bool job_completed;
181
182       simcount_t total_policies;
183       simcount_t checkpoint_skips;
184
185       std::string optimizer_library;
186       arguments_t optimizer_arguments;
187
188       std::string simulator_command;
189       arguments_t simulator_arguments;
190
191       std::string current_map_file;
192       std::string current_lib_file;
193
194       void load( std::istream &in );
195       void save( std::ostream &out );
196
197       void load( const std::string &file );
198       void save( const std::string &file );
199
200       template< class Archive >
201       void serialize( Archive &ar, const unsigned int version );
202     };
203
204   private:
205     void dump_checkpoint( const Job::ptr_t &job, bool job_completed,
206                           simcount_t total_policies, simcount_t
    checkpoint_skips,
207                           const std::string &current_map_file, const std
    ::string &current_lib_file );
208
209     bool restore_checkpoint( const Job::ptr_t &job, bool &job_completed,
210                              simcount_t &total_policies, simcount_t &
    checkpoint_skips,
211                              std::string &current_map_file, std::string
    &current_lib_file );
212
213     bool create_job( const Job::ptr_t &job );
214     void remove_job( const Job::ptr_t &job, JobInfo &job_info );
215
216     void start_dump( const Job::ptr_t &job, JobInfo &job_info, bool
    job_completed );
217     void finish_dump( const Job::ptr_t &job, JobInfo &job_info, bool
    job_completed );
218
219     boost::tuple<bool, unsigned>
220     distribute_policies( const Job::ptr_t &job, JobInfo &job_info, bool
    drop_nodes );
221
222     boost::optional<JobState> manage_init_optimizer( const Job::ptr_t &
    job, JobInfo &job_info );
223     boost::optional<JobState> manage_running_normal( const Job::ptr_t &
    job, JobInfo &job_info );
224     boost::optional<JobState> manage_dumping_states( const Job::ptr_t &
    job, JobInfo &job_info );
225     boost::optional<JobState> manage_dumping_result( const Job::ptr_t &
    job, JobInfo &job_info );
226     boost::optional<JobState> manage( const Job::ptr_t &job, JobInfo &
    job_info );
227
228   private:
229     bool shutdown_;
230     size_t jobs_done_;
231
232     Configuration config_;
233     JobMap<JobInfo, NodeInfo> job_map_;
234     boost::shared_ptr<LogFile> log_file_;
235     boost::shared_ptr<JobQueue> job_queue_;
236   };
237
238   #endif //DICON_MANAGER_MAIN_IMPL_HPP_
```

Listing B.31: src/manager_main_impl.cpp

```
1   /*--- [Distribution] ---------------------------------------------------
2    * This file is part of the Disease Control System DiCon.
3    *
4    * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
5    * Designed and developed with the guidance of Nedialko B. Dimitrov
6    * and Lauren Ancel Meyers at the University of Texas at Austin.
7    *
8    * DiCon is free software: you can redistribute it and/or modify it
9    * under the terms of the GNU General Public License as published by
10   * the Free Software Foundation, either version 3 of the License, or
11   * (at your option) any later version.
12   *
13   * DiCon is distributed in the hope that it will be useful, but
14   * WITHOUT ANY WARRANTY; without even the implied warranty of
15   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
16   * General Public License for more details.
17   *
18   * You should have received a copy of the GNU General Public License
19   * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
20   *-----------------------------------------------------------------*/
21
22  #include "manager_main_impl.hpp"
23  #include "file.hpp"
24  #include <boost/archive/xml_iarchive.hpp>
25  #include <boost/archive/xml_oarchive.hpp>
26  #include <fstream>
27
28
29  namespace detail {
30
31    static const std::string default_config_file       = "main.conf";
32
33    static const std::string default_main_logfile      = "log/main.log";
34    static const std::string default_node_logfile_mask = "log/node_%04u.
    log";
35
36    static const LogLevel default_main_log_level       = LOG_INFO;
37    static const LogLevel default_node_log_level       = LOG_INFO;
38
39
```

```
40    static
41    std::string
42    command_line( const std::string &command, const arguments_t &
      arguments ) {
43      std::string result;
44
45      result += command;
46      BOOST_FOREACH( const std::string &argument, arguments ) {
47        result += '_';
48        result += argument;
49      }
50
51      return result;
52    }
53
54  }
55
56
57
58  MainNodeManagerImpl::JobInfo::JobInfo()
59    : state(JOB_INIT_OPTIMIZER), combined_node(false),
60      optimizer_initd(false), ign_checkpoints(false), getting_policy(
      false),
61      out_of_policy(false), starting_up(true), failure(false),
62      dumping_count(0), total_policies(0), checkpoint_skips(0),
      pending_step_calls(0),
63      checkpoint_time(time_t(-1)), checkpoint_sims(0)
64  {
65  }
66
67
68  MainNodeManagerImpl::NodeInfo::NodeInfo()
69    : simulator_initd(false)
70  {
71  }
72
73
74  MainNodeManagerImpl::MainNodeManagerImpl( boost::mpi::communicator &
      world )
75    : MainNodeManager(world), shutdown_(false), jobs_done_(0)
76  {
77    StatsEntry se( *this, "starting_up", 0 );
78
79    for( int node = min_child(); node <= max_child(); ++node )
80      job_map_.register_node( node );
81
82    // Load main configuration.
83
84    const std::string config_file = detail::default_config_file;
85    config_.import( config_file );
86
87    // Initialize main logfile.
88
89    const std::string &main_logfile   = config_.get( "global", "
      main_logfile"   , detail::default_main_logfile    );
90    LogLevel           main_log_level = config_.get( "global", "
      main_log_level", detail::default_main_log_level );
91
92    log_file_.reset( new LogFile(main_logfile) );
93    log_register_sink( log_file_, main_log_level );
94
95    // Create master's job queue.
96
97    try {
98      job_queue_.reset( new JobQueue(config_) );
99    }
100   catch( boost::exception &e ) {
101     e << errinfo_config_file(config_file);
102     throw;
103   }
104
105   // Initialize node logfiles.
106
107   for( int node = min_child(); node <= max_child(); ++node ) {
108     const std::string &node_logfile_mask = config_.get( "global", "
      node_logfile"   , detail::default_node_logfile_mask );
109     LogLevel           node_log_level    = config_.get( "global", "
      node_log_level", detail::default_node_log_level    );
110     boost::filesystem::path node_logfile;
111
112     {
113       boost::format formatter( node_logfile_mask );
114
115       using namespace boost::io;
116       // Ignore error when mask does not contain a placeholder.
117       formatter.exceptions( all_error_bits ^ too_many_args_bit );
118
119       node_logfile = (formatter % node).str();
120     }
121
122     start_logging( node, node_logfile, node_log_level );
123   }
124 }
125
126
127 boost::filesystem::path
128 MainNodeManagerImpl::job_logfile( const Job::ptr_t &job ) {
129   return job->job_directory / job->job_logfile;
130 }
131
132
133 boost::filesystem::path
134 MainNodeManagerImpl::checkpoint_file( const Job::ptr_t &job ) {
135   return job->job_directory / job->checkpoint_file;
136 }
137
138
139 boost::filesystem::path
140 MainNodeManagerImpl::opt_logfile( const Job::ptr_t &job, int node ) {
141   boost::format formatter( job->opt_logfile_mask );
142
143   using namespace boost::io;
144   // Ignore error when mask does not contain a placeholder.
145   formatter.exceptions( all_error_bits ^ too_many_args_bit );
146
147   return job->job_directory / (formatter % node).str();
148 }
149
150
151 boost::filesystem::path
152 MainNodeManagerImpl::sim_logfile( const Job::ptr_t &job, int node ) {
153   boost::format formatter( job->sim_logfile_mask );
```

```
154
155   using namespace boost::io;
156   // Ignore error when mask does not contain a placeholder.
157   formatter.exceptions( all_error_bits ^ too_many_args_bit );
158
159   return job->job_directory / (formatter % node).str();
160 }
161
162
163 boost::filesystem::path
164 MainNodeManagerImpl::optimizer_map_file( const Job::ptr_t &job,
    simcount_t simulation ) {
165   boost::format formatter( job->optimizer_map_file_mask );
166
167   using namespace boost::io;
168   // Ignore error when mask does not contain a placeholder.
169   formatter.exceptions( all_error_bits ^ too_many_args_bit );
170
171   return job->job_directory / (formatter % simulation).str();
172 }
173
174
175 boost::filesystem::path
176 MainNodeManagerImpl::optimizer_lib_file( const Job::ptr_t &job,
    simcount_t simulation ) {
177   boost::format formatter( job->optimizer_lib_file_mask );
178
179   using namespace boost::io;
180   // Ignore error when mask does not contain a placeholder.
181   formatter.exceptions( all_error_bits ^ too_many_args_bit );
182
183   return job->job_directory / (formatter % simulation).str();
184 }
185
186
187 boost::filesystem::path
188 MainNodeManagerImpl::policy_bin_dumpfile( const Job::ptr_t &job,
    simcount_t simulation ) {
189   boost::format formatter( job->policy_bin_dumpfile_mask );
190
191   using namespace boost::io;
192   // Ignore error when mask does not contain a placeholder.
193   formatter.exceptions( all_error_bits ^ too_many_args_bit );
194
195   return job->job_directory / (formatter % simulation).str();
196 }
197
198
199 boost::filesystem::path
200 MainNodeManagerImpl::policy_txt_dumpfile( const Job::ptr_t &job,
    simcount_t simulation ) {
201   boost::format formatter( job->policy_txt_dumpfile_mask );
202
203   using namespace boost::io;
204   // Ignore error when mask does not contain a placeholder.
205   formatter.exceptions( all_error_bits ^ too_many_args_bit );
206
207   return job->job_directory / (formatter % simulation).str();
208 }
209
```

```
210
211 boost::filesystem::path
212 MainNodeManagerImpl::policy_bin_result( const Job::ptr_t &job ) {
213   return job->job_directory / job->policy_bin_result;
214 }
215
216
217 boost::filesystem::path
218 MainNodeManagerImpl::policy_txt_result( const Job::ptr_t &job ) {
219   return job->job_directory / job->policy_txt_result;
220 }
221
222
223 void
224 MainNodeManagerImpl::finish_start_logging( int node ) {
225   // Nothing to do.
226 }
227
228
229 void
230 MainNodeManagerImpl::finish_init_optimizer( int node ) {
231   const Job::ptr_t &job = job_map_.job( node );
232   JobInfo &job_info = job_map_.job_info( job );
233   assert( !job_info.combined_node );
234
235   assert( !job_info.optimizer_initd );
236   job_info.optimizer_initd = true;
237 }
238
239
240 void
241 MainNodeManagerImpl::finish_init_simulator( int node ) {
242   const Job::ptr_t &job = job_map_.job( node );
243   JobInfo &job_info = job_map_.job_info( job );
244   assert( !job_info.combined_node );
245   (void) job_info;
246
247   NodeInfo &node_info = job_map_.node_info( job, node );
248
249   assert( !node_info.simulator_initd );
250   node_info.simulator_initd = true;
251 }
252
253
254 void
255 MainNodeManagerImpl::finish_init_combined( int node ) {
256   const Job::ptr_t &job = job_map_.job( node );
257   JobInfo &job_info = job_map_.job_info( job );
258   assert( job_info.combined_node );
259
260   assert( !job_info.optimizer_initd );
261   job_info.optimizer_initd = true;
262 }
263
264
265 void
266 MainNodeManagerImpl::finish_get_policy( int node, const boost::
    optional<policy_t> &policy ) {
267   const Job::ptr_t &job = job_map_.job( node );
268   JobInfo &job_info = job_map_.job_info( job );
269   assert( !job_info.combined_node );
```

```
270
271   assert( job_info.getting_policy );
272   job_info.getting_policy = false;
273
274   assert( !job_info.out_of_policy );
275
276   if( policy ) {
277     job_info.backlog.push( *policy );
278     ++job_info.total_policies;
279   }
280   else
281     job_info.out_of_policy = true;
282 }
283
284
285 void
286 MainNodeManagerImpl::finish_simulate( int node, double reward ) {
287   const Job::ptr_t &job = job_map_.job( node );
288   JobInfo &job_info = job_map_.job_info( job );
289   assert( !job_info.combined_node );
290
291   int opt_node = job_map_.optimizer_node( job );
292   NodeInfo &node_info = job_map_.node_info( job, node );
293
294   assert( !node_info.simulate_queue.empty() );
295   policy_t policy = node_info.simulate_queue.front();
296   node_info.simulate_queue.pop();
297
298   update( opt_node, policy, reward );
299
300   job_info.update_queue.push( policy );
301 }
302
303
304 void
305 MainNodeManagerImpl::finish_update( int node ) {
306   const Job::ptr_t &job = job_map_.job( node );
307   JobInfo &job_info = job_map_.job_info( job );
308   assert( !job_info.combined_node );
309
310   assert( !job_info.update_queue.empty() );
311   job_info.update_queue.pop();
312
313   job_info.out_of_policy = false;
314 }
315
316
317 void
318 MainNodeManagerImpl::finish_step_combined( int node, bool got_policy )
    {
319   const Job::ptr_t &job = job_map_.job( node );
320   JobInfo &job_info = job_map_.job_info( job );
321   assert( job_info.combined_node );
322
323   assert( job_info.pending_step_calls > 0 );
324   --job_info.pending_step_calls;
325
326   if( !got_policy ) {
327     --job_info.total_policies;
328     job_info.out_of_policy = true;
329   }
330 }
331
332
333 void
334 MainNodeManagerImpl::finish_dump_optimizer( int node ) {
335   const Job::ptr_t &job = job_map_.job( node );
336   JobInfo &job_info = job_map_.job_info( job );
337
338   assert( job_info.dumping_count > 1 );
339   --job_info.dumping_count;
340 }
341
342
343 void
344 MainNodeManagerImpl::finish_dump_policies( int node ) {
345   const Job::ptr_t &job = job_map_.job( node );
346   JobInfo &job_info = job_map_.job_info( job );
347
348   assert( job_info.dumping_count > 1 );
349   --job_info.dumping_count;
350 }
351
352
353 void
354 MainNodeManagerImpl::finish_shutdown( int node, bool done ) {
355   assert( done );
356 }
357
358
359 void
360 MainNodeManagerImpl::handle_failure_without_job( int node, const std::
    string &what ) {
361   if( state(node) == NODE_SHUTTING_DOWN ) {
362     // Failed to shutdown node gracefully. Throw exception to force
363     // the system down, by indirectly calling MPI_Abort in main().
364
365     log_failure( (boost::format("Failed to shutdown at node #%u. 
    Aborting system.\n%s")
366                   % node % what).str() );
367
368     BOOST_THROW_EXCEPTION( NodeManagerShutdownError() );
369   }
370
371   if( !shutdown_ ) {
372     log_failure( (boost::format("Request (%s) failed at node #%u. 
    Stopping system.\n%s")
373                   % boost::lexical_cast<std::string>(state(node)) % 
    node % what).str() );
374
375     shutdown_ = true;
376   }
377 }
378
379
380 void
381 MainNodeManagerImpl::handle_failure_with_job( int node, const std::
    string &what ) {
382   const Job::ptr_t &job = job_map_.job( node );
383   JobInfo &job_info = job_map_.job_info( job );
384
385   if( !job_info.failure ) {
```

```
386       log_error( job->id,
387                  (boost::format("Request (%s) failed at node #%u. Stopping job.\n%s")
388                   % boost::lexical_cast<std::string>(state(node)) % node % what).str() );
389
390       job_info.failure = true;
391     }
392   }
393
394
395   void
396   MainNodeManagerImpl::handle_failure( int node, const std::string &what ) {
397     switch( state(node) ) {
398     case NODE_IDLE:
399       BOOST_THROW_EXCEPTION( AssertionError() );
400       break;
401
402     case NODE_STARTING_LOGGING:
403     case NODE_SHUTTING_DOWN:
404       handle_failure_without_job( node, what );
405       break;
406
407     case NODE_INITIALIZING_OPTIMIZER:
408     case NODE_INITIALIZING_SIMULATOR:
409     case NODE_INITIALIZING_COMBINED:
410     case NODE_GETTING_POLICY:
411     case NODE_SIMULATING:
412     case NODE_UPDATING:
413     case NODE_STEPPING_COMBINED:
414     case NODE_DUMPING_OPTIMIZER:
415     case NODE_DUMPING_POLICIES:
416       handle_failure_with_job( node, what );
417       break;
418     }
419   }
420
421
422   template< class Archive >
423   void
424   MainNodeManagerImpl::Checkpoint::serialize( Archive &ar, const unsigned int version ) {
425     ar & boost::serialization::make_nvp( "job_completed"      , job_completed      );
426
427     ar & boost::serialization::make_nvp( "total_policies"     , total_policies     );
428     ar & boost::serialization::make_nvp( "checkpoint_skips"   , checkpoint_skips   );
429
430     ar & boost::serialization::make_nvp( "optimizer_library"  , optimizer_library  );
431     ar & boost::serialization::make_nvp( "optimizer_arguments", optimizer_arguments );
432
433     ar & boost::serialization::make_nvp( "simulator_command"  , simulator_command  );
434     ar & boost::serialization::make_nvp( "simulator_arguments", simulator_arguments );
435
436     ar & boost::serialization::make_nvp( "current_map_file"   , current_map_file   );
437     ar & boost::serialization::make_nvp( "current_lib_file"   , current_lib_file   );
438   }
439
440
441   void
442   MainNodeManagerImpl::Checkpoint::load( std::istream &in ) {
443     boost::archive::xml_iarchive( in )
444       >> boost::serialization::make_nvp( "checkpoint", *this );
445   }
446
447
448   void
449   MainNodeManagerImpl::Checkpoint::save( std::ostream &out ) {
450     boost::archive::xml_oarchive( out )
451       << boost::serialization::make_nvp( "checkpoint", *this );
452   }
453
454
455   void
456   MainNodeManagerImpl::Checkpoint::load( const std::string &file ) {
457     std::ifstream in( file.c_str() );
458     if( in.fail() )
459       BOOST_THROW_EXCEPTION( FileReadError() << errinfo_file_name(file) );
460     load( in );
461   }
462
463
464   void
465   MainNodeManagerImpl::Checkpoint::save( const std::string &file ) {
466     std::ofstream out( file.c_str() );
467     save( out );
468     if( out.fail() )
469       BOOST_THROW_EXCEPTION( FileWriteError() << errinfo_file_name(file) );
470   }
471
472
473   void
474   MainNodeManagerImpl::dump_checkpoint( const Job::ptr_t &job, bool job_completed,
475                                         simcount_t total_policies, simcount_t checkpoint_skips,
476                                         const std::string &current_map_file, const std::string &current_lib_file )
477   {
478     const boost::filesystem::path &file = checkpoint_file( job );
479
480     Checkpoint checkpoint;
481
482     checkpoint.optimizer_library   = job->optimizer_library;
483     checkpoint.optimizer_arguments = job->optimizer_arguments;
484     checkpoint.simulator_command   = job->simulator_command;
485     checkpoint.simulator_arguments = job->simulator_arguments;
486
487     checkpoint.job_completed    = job_completed;
488     checkpoint.total_policies   = total_policies;
489     checkpoint.checkpoint_skips = checkpoint_skips;
```

```
490     checkpoint.current_map_file = current_map_file;
491     checkpoint.current_lib_file = current_lib_file;
492
493     boost::filesystem::path tmp_file = File::unique_temporary( file );
494
495     checkpoint.save( tmp_file.file_string() );
496
497     File::rename( tmp_file, file );
498
499     log_info( job->id,
500               (boost::format("Wrote checkpoint file '%s'.") % file.file_string()).str() );
501 }
502
503
504 bool
505 MainNodeManagerImpl::restore_checkpoint( const Job::ptr_t &job, bool &job_completed,
506                                          simcount_t &total_policies, simcount_t &checkpoint_skips,
507                                          std::string &current_map_file, std::string &current_lib_file )
508 {
509   const boost::filesystem::path &file = checkpoint_file( job );
510
511   if( !File::exists(file) ) {
512     log_info( job->id,
513               (boost::format("Checkpoint file '%s' does not exist.") % file.file_string()).str() );
514     return false;
515   }
516
517   Checkpoint checkpoint;
518   checkpoint.load( file.file_string() );
519
520   log_info( job->id,
521             (boost::format("Read checkpoint data from file '%s'.") % file.file_string()).str() );
522
523   if( checkpoint.optimizer_library    != job->optimizer_library    ||
524       checkpoint.optimizer_arguments  != job->optimizer_arguments  ||
525       checkpoint.simulator_command    != job->simulator_command    ||
526       checkpoint.simulator_arguments  != job->simulator_arguments )
527   {
528     BOOST_THROW_EXCEPTION( CheckpointNoMatchError() << errinfo_file_name(file.file_string()) );
529   }
530
531   job_completed    = checkpoint.job_completed;
532   total_policies   = checkpoint.total_policies;
533   checkpoint_skips = checkpoint.checkpoint_skips;
534   current_map_file = checkpoint.current_map_file;
535   current_lib_file = checkpoint.current_lib_file;
536
537   return true;
538 }
539
540
541 boost::optional<MainNodeManagerImpl::JobState>
542 MainNodeManagerImpl::manage_init_optimizer( const Job::ptr_t &job, JobInfo &job_info ) {
543   // Check if optimizer is ready now.
544
545   if( job_info.optimizer_initd ) {
546     if( (job_info.checkpoint_time = time(NULL)) == time_t(-1) )
547       BOOST_THROW_EXCEPTION( TimeError() << errinfo_api_function("time") << errinfo_errno(errno) );
548
549     job_info.checkpoint_sims = job_info.total_policies;
550
551     log_info( job->id,
552               "Optimizer has initialized. Starting simulation." );
553
554     return JOB_RUNNING_NORMAL;
555   }
556
557   return boost::none;
558 }
559
560
561 boost::tuple<bool, unsigned>
562 MainNodeManagerImpl::distribute_policies( const Job::ptr_t &job, JobInfo &job_info, bool drop_nodes ) {
563   bool sim_initializing = false;
564   unsigned total_sim_queue = 0;
565
566   // Check each simulator node: if simulator queue not full and policy
567   // available, push another policy on queue; if simulator queue empty
568   // and no policies available, remove node from current job (a max of
569   // one node will be removed per invocation, and only if at least two
570   // nodes are idling).
571
572   typedef std::multimap<size_t, int> queues_t;
573   queues_t queues;
574
575   // Generate list of nodes that are candidates for new policies. Also
576   // save current backlog size for each node. Sort nodes, according to
577   // backlog size, in multimap.
578
579   BOOST_FOREACH( int sim_node, job_map_.simulator_nodes(job) ) {
580     NodeInfo &node_info = job_map_.node_info( job, sim_node );
581
582     total_sim_queue += node_info.simulate_queue.size();
583
584     if( !node_info.simulator_initd ) {
585       sim_initializing = true;
586       continue;
587     }
588
589     queues.insert( std::make_pair(node_info.simulate_queue.size(), sim_node) );
590   }
591
592   // As long as jobs are in job backlog, and nodes are available, push
593   // policies to node with smallest number of policies in its backlog.
594
595   while( !job_info.backlog.empty() ) {
596     const policy_t &policy = job_info.backlog.front();
597
598     if( queues.empty() )
599       break;
600
```

```
601     queues_t::iterator it = queues.begin();
602
603     size_t size = it->first;
604     int sim_node = it->second;
605
606     if( !(size < (job->node_backlog + 1)) )
607       break;
608
609     NodeInfo &node_info = job_map_.node_info( job, sim_node );
610     assert( node_info.simulate_queue.size() == size );
611
612     simulate( sim_node, policy );
613
614     node_info.simulate_queue.push( policy );
615     ++total_sim_queue;
616
617     queues.erase( it );
618     assert( node_info.simulate_queue.size() == size+1 );
619     queues.insert( std::make_pair(size+1, sim_node) );
620
621     job_info.backlog.pop();
622   }
623
624   // If at least two nodes are running empty, and no more policies are
625   // available right now, drop one of those nodes from the job.
626
627   if( drop_nodes && job_info.out_of_policy
628       && queues.size() >= 2 && queues.begin()->first == 0
629       && (++queues.begin())->first == 0 && job_info.backlog.empty() )
630   {
631     int sim_node = queues.begin()->second;
632
633     assert( state(sim_node) == NODE_IDLE );
634
635     log_info( job->id,
636               (boost::format("Not enough policies available. Removing ↩
    ↪simulator node #%u from job.") % sim_node).str() );
637
638     job_map_.release_node( job, sim_node );
639   }
640
641   return boost::make_tuple( sim_initializing, total_sim_queue );
642 }
643
644
645 boost::optional<MainNodeManagerImpl::JobState>
646 MainNodeManagerImpl::manage_running_normal( const Job::ptr_t &job, ↩
    ↪JobInfo &job_info ) {
647   bool repeat_this = false;
648   bool reached_max_sims = false;
649
650   if( job->max_sims != 0 && job_info.total_policies >= job->max_sims ) ↩
    ↪{
651     reached_max_sims = true;
652   }
653
654   if( !job_info.combined_node ) {
655     // If no policies are available anymore, job is started. This ↩
    ↪allows
656     // a new job to be created in run() below.
657
658     if( job_info.out_of_policy )
659       job_info.starting_up = false;
660   }
661   else {
662     // Combined opt/ sim node. Job is always "started" to allow new ↩
    ↪jobs
663     // to be created in run(). For combined nodes, the maximum number ↩
    ↪of
664     // nodes has been reached from the beginning, anyway.
665
666     job_info.starting_up = false;
667   }
668
669   // These variables will only be valid for job_info.combined_node ↩
    ↪equal
670   // to false (they are not needed for combined opt/ sim nodes, anyway↩
    ↪).
671   bool sim_initializing = false;
672   unsigned total_sim_queue = 0;
673
674   if( !job_info.combined_node ) {
675     // Distribute policies among simulator nodes. Remove nodes which ↩
    ↪are
676     // idling (if no policies are available for distribution anymore).
677
678     boost::tie(sim_initializing, total_sim_queue)
679       = distribute_policies( job, job_info, true );
680
681     // If there are still policies available in backlog and job can ↩
    ↪take
682     // another node, assign available node to job; initialize ↩
    ↪simulator.
683
684     if( !job_info.backlog.empty() && !sim_initializing && job_map_.↩
    ↪free_node_count() != 0 &&
685         (job->max_nodes == 0 || job_map_.simulator_nodes(job).size() <↩
    ↪ job->max_nodes) )
686     {
687       int sim_node = job_map_.get_free_node();
688       assert( state(sim_node) == NODE_IDLE );
689
690       log_info( job->id,
691                 (boost::format("All simulators are busy. Initializing ↩
    ↪new simulator on node #%u.") % sim_node).str() );
692
693       init_simulator( sim_node
694                     , sim_logfile(job, sim_node)
695                     , job->simulator_command, job->simulator_arguments
696                     );
697
698       job_map_.assign_node( job, sim_node );
699     }
700
701     // If maximum number of nodes has been reached, job is started. ↩
    ↪This
702     // allows a new job to be created in run() below.
703
704     if( job->max_nodes != 0 && job_map_.simulator_nodes(job).size() >=↩
    ↪ job->max_nodes )
705       job_info.starting_up = false;
706   }
```

```
707     else {
708       // Combined opt/ sim node. Step only one policy at a time because we
709       // don't want to miss the checkpoint condition (checked below); loop
710       // this implicitly by returning the JOB_RUNNING_NORMAL state again.
711
712       if( !reached_max_sims && !job_info.out_of_policy &&
713           job_info.pending_step_calls < (job->node_backlog + 1) )
714       {
715         step_combined( job_map_.optimizer_node(job) );
716         ++job_info.pending_step_calls;
717         ++job_info.total_policies;
718         repeat_this = true;
719       }
720     }
721
722     // Check to see if we have reached a checkpoint. This happens either
723     // when the desired number of policies (i.e., simulations) since the
724     // last checkpoint has been reached or the desired number of seconds
725     // since the last checkpoint has passed.
726
727     bool reached_checkpoint = false;
728
729     time_t current_time;
730     if( (current_time = time(NULL)) == time_t(-1) )
731       BOOST_THROW_EXCEPTION( TimeError() << errinfo_api_function("time") << errinfo_errno(errno) );
732
733     if(  (job->checkpoint_sims != 0 && unsigned(job_info.total_policies - job_info.checkpoint_sims) >= job->checkpoint_sims)
734       || (job->checkpoint_secs != 0 && unsigned(current_time - job_info.checkpoint_time) >= job->checkpoint_secs)
735       )
736     {
737       reached_checkpoint = true;
738     }
739
740     if( job->checkpoints->has() &&
741         !job_info.ign_checkpoints &&
742         job_info.total_policies != job_info.checkpoint_sims )
743     {
744       boost::optional<simcount_t> checkpoint;
745
746       try {
747         checkpoint = job->checkpoints->get();
748       }
749       catch( boost::exception &e ) {
750         log_error( job->id,
751                    (boost::format("Error in checkpoint_list. Disabling (nonrecurring) checkpoints.\n%s")
752                     % boost::diagnostic_information(e)).str() );
753         job_info.ign_checkpoints = true;
754       }
755
756       if( checkpoint && *checkpoint == job_info.total_policies ) {
757         ++job_info.checkpoint_skips;
758         reached_checkpoint = true;
759         job->checkpoints->inc();
760       }
761       else if( checkpoint && *checkpoint < job_info.total_policies ) {
762         log_error( job->id,
763                    (boost::format("Checkpoint list is not a strictly increasing sequence of positive numbers. "
764                                   "(got '%u' in checkpoint_list after doing %u simulations). "
765                                   "Disabling (nonrecurring) checkpoints.")
766                     % *checkpoint % job_info.total_policies).str() );
767         job_info.ign_checkpoints = true;
768       }
769     }
770
771     if( !job_info.combined_node ) {
772       // When not currently getting a policy and we don't know for sure we
773       // are out of policies, try to get a new policy; if backlog permits,
774       // and we are not doing a checkpoint next or reached the max. number
775       // of simulations.
776
777       if( !job_info.getting_policy && !job_info.out_of_policy &&
778           job_info.backlog.size() < (job->job_backlog + 1) &&
779           !reached_checkpoint && !reached_max_sims )
780       {
781         int opt_node = job_map_.optimizer_node( job );
782         job_info.getting_policy = true;
783         get_policy( opt_node );
784       }
785
786       // When there are no more policies available, and no simulator works
787       // on any simulation right now, we have reached the end of this job.
788
789       if( total_sim_queue == 0 &&
790           (reached_max_sims || job_info.out_of_policy) &&
791           job_info.backlog.empty() && job_info.update_queue.empty() )
792       {
793         BOOST_FOREACH( int sim_node, job_map_.simulator_nodes(job) ) {
794           assert( state(sim_node) == NODE_IDLE );
795
796           job_map_.release_node( job, sim_node );
797         }
798
799         return JOB_DUMPING_RESULT;
800       }
801     }
802     else {
803       // Combined opt/ sim node. When step_combined() returned false once,
804       // and all remaining step_combined() calls have returned, we reached
805       // the end the job.
806
807       if( (reached_max_sims || job_info.out_of_policy) &&
808           job_info.pending_step_calls == 0 )
809       {
810         return JOB_DUMPING_RESULT;
811       }
```

```
812     }
813
814     // If we are to have a checkpoint, allow other jobs to start now (we
815     // won't be using any more nodes while checkpointing), and go change
816     // the state accordingly.
817
818     if( reached_checkpoint && !reached_max_sims ) {
819       job_info.starting_up = false;
820
821       return JOB_DUMPING_STATES;
822     }
823
824     if( !repeat_this )
825       return boost::none;
826     else
827       return JOB_RUNNING_NORMAL;
828   }
829
830
831   void
832   MainNodeManagerImpl::start_dump( const Job::ptr_t &job, JobInfo &
      job_info, bool job_completed ) {
833     assert( job_info.dumping_count == 0 );
834     job_info.dumping_count = 1;
835
836     job_info.current_map_file.clear();
837     job_info.current_lib_file.clear();
838
839     int opt_node = job_map_.optimizer_node( job );
840
841     boost::filesystem::path optimizer_map_file  = this->
      optimizer_map_file( job, job_info.total_policies );
842     boost::filesystem::path optimizer_lib_file  = this->
      optimizer_lib_file( job, job_info.total_policies );
843     boost::filesystem::path policy_bin_dumpfile = this->
      policy_bin_dumpfile( job, job_info.total_policies );
844     boost::filesystem::path policy_txt_dumpfile = this->
      policy_txt_dumpfile( job, job_info.total_policies );
845
846     if( !optimizer_map_file.empty() ) {
847       optimizer_map_file = File::unique_name( optimizer_map_file );
848       log_info( job->id,
849                 (boost::format("Writing optimizer map to '%s'.") %
      optimizer_map_file.file_string()).str() );
850       job_info.current_map_file = optimizer_map_file.file_string();
851       dump_optimizer( opt_node, optimizer_map_file, DUMP_OPTIMIZER_MAP )
      ;
852       ++job_info.dumping_count;
853     }
854
855     if( !optimizer_lib_file.empty() ) {
856       optimizer_lib_file = File::unique_name( optimizer_lib_file );
857       log_info( job->id,
858                 (boost::format("Writing optimizer state to '%s'.") %
      optimizer_lib_file.file_string()).str() );
859       job_info.current_lib_file = optimizer_lib_file.file_string();
860       dump_optimizer( opt_node, optimizer_lib_file, DUMP_OPTIMIZER_LIB )
      ;
861       ++job_info.dumping_count;
862     }
863
```

```
864     if( !policy_bin_dumpfile.empty() ) {
865       policy_bin_dumpfile = File::unique_name( policy_bin_dumpfile );
866       log_info( job->id,
867                 (boost::format("Writing best policy list (binary) to '%s
      '.") % policy_bin_dumpfile.file_string()).str() );
868       dump_policies( opt_node, policy_bin_dumpfile, job->policy_count,
      false );
869       ++job_info.dumping_count;
870     }
871
872     if( !policy_txt_dumpfile.empty() ) {
873       policy_txt_dumpfile = File::unique_name( policy_txt_dumpfile );
874       log_info( job->id,
875                 (boost::format("Writing best policy list (textual) to '%
      s'.") % policy_txt_dumpfile.file_string()).str() );
876       dump_policies( opt_node, policy_txt_dumpfile, job->policy_count,
      true );
877       ++job_info.dumping_count;
878     }
879
880     if( job_completed ) {
881       boost::filesystem::path policy_bin_result = this->
      policy_bin_result( job );
882       boost::filesystem::path policy_txt_result = this->
      policy_txt_result( job );
883
884       if( !policy_bin_result.empty() ) {
885         policy_bin_result = File::unique_name( policy_bin_result );
886         log_info( job->id,
887                   (boost::format("Writing final best policy list (binary
      ) to '%s'.") % policy_bin_result.file_string()).str() );
888         dump_policies( opt_node, policy_bin_result, job->policy_count,
      false );
889         ++job_info.dumping_count;
890       }
891
892       if( !policy_txt_result.empty() ) {
893         policy_txt_result = File::unique_name( policy_txt_result );
894         log_info( job->id,
895                   (boost::format("Writing final best policy list (
      textual) to '%s'.") % policy_txt_result.file_string()).str() );
896         dump_policies( opt_node, policy_txt_result, job->policy_count,
      true );
897         ++job_info.dumping_count;
898       }
899     }
900   }
901
902
903   void
904   MainNodeManagerImpl::finish_dump( const Job::ptr_t &job, JobInfo &
      job_info, bool job_completed ) {
905     assert( job_info.dumping_count == 1 );
906     job_info.dumping_count = 0;
907
908     try {
909       dump_checkpoint( job, job_completed,
910                        job_info.total_policies, job_info.
      checkpoint_skips,
911                        job_info.current_map_file, job_info.
      current_lib_file );
```

```
912     }
913     catch( boost::exception &e ) {
914       log_error( job->id,
915                  (boost::format("Failed to dump checkpoint.\n%s") % ↩
    ↪ diagnostic_information(e)).str() );
916     }
917
918     if( !job_completed ) {
919       if( (job_info.checkpoint_time = time(NULL)) == time_t(-1) )
920         BOOST_THROW_EXCEPTION( TimeError() << errinfo_api_function("time↩
    ↪") << errinfo_errno(errno) );
921
922       job_info.checkpoint_sims = job_info.total_policies;
923     }
924   }
925
926
927   boost::optional<MainNodeManagerImpl::JobState>
928   MainNodeManagerImpl::manage_dumping_states( const Job::ptr_t &job, ↩
    ↪ JobInfo &job_info ) {
929     if( job_info.dumping_count == 0 ) {
930       bool start_dumping = false;
931
932       if( !job_info.combined_node ) {
933         // Distribute policies among simulator nodes but do NOT remove ↩
    ↪ nodes
934         // that are idling: we will need them again once state dump has ↩
    ↪ been
935         // completed.
936
937         unsigned total_sim_queue;
938
939         boost::tie(boost::tuples::ignore, total_sim_queue)
940           = distribute_policies( job, job_info, false );
941
942         // As soon as all policies from policy backlog have been done, ↩
    ↪ start
943         // the dumping, according to job's config settings.
944
945         if( total_sim_queue == 0 &&
946             job_info.backlog.empty() )
947         {
948           start_dumping = true;
949         }
950       }
951       else {
952         // Combined opt/ sim node. As soon as no step_combined() calls ↩
    ↪ are
953         // active anymore, start the actual dumping.
954
955         if( job_info.out_of_policy && job_info.pending_step_calls == 0 )↩
    ↪ {
956           // The step_combined() function returned false, i.e., we ↩
    ↪ reached
957           // the end of the job. Therefore this state dump isn't ↩
    ↪ necessary
958           // anymore and we can directly proceed to dumping the results.
959
960           return JOB_DUMPING_RESULT;
961         }
962         else if( job_info.pending_step_calls == 0 )
963           start_dumping = true;
964       }
965
966       if( start_dumping ) {
967         log_info( job->id,
968                   (boost::format("Creating checkpoint after %u ↩
    ↪ simulations.") % job_info.total_policies).str() );
969
970         start_dump( job, job_info, false );
971         assert( job_info.dumping_count >= 1 );
972       }
973     }
974
975     // When all dump operations have finished, write checkpoint file, ↩
    ↪ then
976     // reset job to normal mode.
977
978     if( job_info.dumping_count == 1 ) {
979       finish_dump( job, job_info, false );
980       assert( job_info.dumping_count == 0 );
981
982       log_info( job->id,
983                 "Finished creating checkpoint." );
984
985       return JOB_RUNNING_NORMAL;
986     }
987
988     return boost::none;
989   }
990
991
992   boost::optional<MainNodeManagerImpl::JobState>
993   MainNodeManagerImpl::manage_dumping_result( const Job::ptr_t &job, ↩
    ↪ JobInfo &job_info ) {
994     if( job_info.dumping_count == 0 ) {
995       log_info( job->id,
996                 (boost::format("Reached end of job after %u simulations.↩
    ↪ Dumping results.") % job_info.total_policies).str() );
997
998       start_dump( job, job_info, true );
999       assert( job_info.dumping_count >= 1 );
1000    }
1001
1002    // When all dump operations have finished, finish job and free node.
1003
1004    if( job_info.dumping_count == 1 ) {
1005      finish_dump( job, job_info, true );
1006      assert( job_info.dumping_count == 0 );
1007
1008      int opt_node = job_map_.optimizer_node( job );
1009      assert( state(opt_node) == NODE_IDLE );
1010
1011      log_info( job->id,
1012                "Finished dumping results. Stopping job." );
1013
1014      job_map_.finish_job( job, opt_node );
1015
1016      ++jobs_done_;
1017
1018      return boost::none;
1019    }
```

```
1020
1021    return boost::none;
1022  }
1023
1024
1025  boost::optional<MainNodeManagerImpl::JobState>
1026  MainNodeManagerImpl::manage( const Job::ptr_t &job, JobInfo &job_info ) {
1027    switch( job_info.state ) {
1028    case JOB_INIT_OPTIMIZER: return manage_init_optimizer(job, job_info); break;
1029    case JOB_RUNNING_NORMAL: return manage_running_normal(job, job_info); break;
1030    case JOB_DUMPING_STATES: return manage_dumping_states(job, job_info); break;
1031    case JOB_DUMPING_RESULT: return manage_dumping_result(job, job_info); break;
1032    }
1033
1034    BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<JobState>(job_info.state) );
1035  }
1036
1037
1038  bool
1039  MainNodeManagerImpl::create_job( const Job::ptr_t &job ) {
1040    boost::shared_ptr<LogFile> job_logfile( new LogFile(this->job_logfile(job)) );
1041    log_register_sink( job_logfile, job->job_log_level, job->id );
1042
1043    log_info( job->id,
1044              ( boost::format("Starting job (#%u; job=%u, argument=%u) in directory '%s'."
1045                              "\nOptimizer command line: '%s'\nSimulator command line: '%s'")
1046              % job->id % job->job_set_id % job->arg_set_id % job->job_directory.file_string()
1047              % detail::command_line(job->optimizer_library, job->optimizer_arguments)
1048              % detail::command_line(job->simulator_command, job->simulator_arguments)
1049              ).str() );
1050
1051    bool restore_job = false;
1052
1053    bool job_completed;
1054    simcount_t total_policies;
1055    simcount_t checkpoint_skips;
1056    std::string current_map_file;
1057    std::string current_lib_file;
1058
1059    bool got_checkpoint;
1060
1061    try {
1062      got_checkpoint = restore_checkpoint( job, job_completed,
1063                                           total_policies, checkpoint_skips,
1064                                           current_map_file, current_lib_file );
1065    }
1066    catch( boost::exception &e ) {
1067      log_error( job->id,
1068                 (boost::format("Failed to restore checkpoint. Skipping job.\n%s") % diagnostic_information(e)).str() );
1069      return false;
1070    }
1071
1072    if( got_checkpoint ) {
1073      if( job_completed ) {
1074        log_info( job->id,
1075                  "Job has checkpoint; has already completed. Skipping job." );
1076        ++jobs_done_;
1077        return false;
1078      }
1079
1080      // Check if we actually have a dumpfile: dumping the optimizer state
1081      // might have been disabled at the time the checkpoint was taken.
1082
1083      if( !current_map_file.empty() && !current_lib_file.empty() ) {
1084        restore_job = true;
1085      }
1086    }
1087
1088    if( restore_job ) {
1089      log_info( job->id,
1090                (boost::format("Job has checkpoint; continuing job with %u simulations done.") % total_policies).str() );
1091    }
1092    else {
1093      log_info( job->id,
1094                "Job has no checkpoints; beginning job from scratch." );
1095    }
1096
1097    const bool create_combined = job->max_nodes == 1;
1098
1099    int opt_node = job_map_.get_free_node();
1100    assert( state(opt_node) == NODE_IDLE );
1101
1102    if( !create_combined ) {
1103      // Use regular optimizer node.
1104
1105      log_info( job->id,
1106                (boost::format("Initializing optimizer on node #%u.") % opt_node).str() );
1107
1108      init_optimizer( opt_node
1109                    , opt_logfile(job, opt_node)
1110                    , sim_logfile(job, opt_node)
1111                    , job->optimizer_library, job->optimizer_arguments
1112                    , job->simulator_command, job->simulator_arguments
1113                    , (restore_job ? boost::optional<boost::filesystem::path>(current_map_file) : boost::none)
1114                    , (restore_job ? boost::optional<boost::filesystem::path>(current_lib_file) : boost::none)
1115                    );
1116    }
1117    else {
1118      // Use combined opt/ sim node.
1119
1120      log_info( job->id,
```

```
1121                (boost::format("Initializing combined optimizer/
    simulator on node #%u.") % opt_node).str() );
1122
1123      init_combined( opt_node
1124                   , opt_logfile(job, opt_node)
1125                   , sim_logfile(job, opt_node)
1126                   , job->optimizer_library, job->optimizer_arguments
1127                   , job->simulator_command, job->simulator_arguments
1128                   , (restore_job ? boost::optional<boost::filesystem::
    path>(current_map_file) : boost::none)
1129                   , (restore_job ? boost::optional<boost::filesystem::
    path>(current_lib_file) : boost::none)
1130                   );
1131    }
1132
1133    job_map_.create_job( job, opt_node );
1134
1135    JobInfo &job_info = job_map_.job_info( job );
1136    job_info.job_logfile = job_logfile;
1137
1138    if( !create_combined ) {
1139      int sim_node = job_map_.get_free_node();
1140      assert( state(sim_node) == NODE_IDLE );
1141
1142      log_info( job->id,
1143                (boost::format("Initializing simulator on node #%u.") %
    sim_node).str() );
1144
1145      init_simulator( sim_node
1146                    , sim_logfile(job, sim_node)
1147                    , job->simulator_command, job->simulator_arguments
1148                    );
1149
1150      job_map_.assign_node( job, sim_node );
1151    }
1152    else
1153      job_info.combined_node = true;
1154
1155    if( restore_job ) {
1156      job_info.total_policies = total_policies;
1157      job_info.checkpoint_skips = checkpoint_skips;
1158
1159      for( simcount_t i = 0; i < job_info.checkpoint_skips && job->
    checkpoints->has(); ++i ) {
1160        job->checkpoints->inc();
1161      }
1162    }
1163
1164    return true;
1165  }
1166
1167
1168  void
1169  MainNodeManagerImpl::remove_job( const Job::ptr_t &job, JobInfo &
    job_info ) {
1170    BOOST_FOREACH( int sim_node, job_map_.simulator_nodes(job) ) {
1171      if( state(sim_node) == NODE_IDLE )
1172        job_map_.release_node( job, sim_node );
1173    }
1174
1175    int opt_node = job_map_.optimizer_node( job );
1176
1177    if( state(opt_node) == NODE_IDLE &&
1178        job_map_.simulator_nodes(job).empty() )
1179    {
1180      job_map_.finish_job( job, opt_node );
1181    }
1182  }
1183
1184
1185  void
1186  MainNodeManagerImpl::run() {
1187    StatsEntry se_running( *this, "running", 50 );
1188
1189    // Wait for start_logging() to complete.
1190
1191    while( process() );
1192
1193    // Main loop: process job state machines.
1194
1195    do {
1196      StatsEntry se_scheduling( *this, "scheduling", 50 );
1197
1198      if( shutdown_ ) {
1199        // Shutting down: process and finish only remaining jobs.
1200
1201        BOOST_FOREACH( const Job::ptr_t &job, job_map_.jobs() ) {
1202          JobInfo &job_info = job_map_.job_info( job );
1203
1204          // Handle cleaning-up of job.
1205          remove_job( job, job_info );
1206        }
1207
1208        continue;
1209      }
1210
1211      bool starting_up = false;
1212
1213      BOOST_FOREACH( const Job::ptr_t &job, job_map_.jobs() ) {
1214        JobInfo &job_info = job_map_.job_info( job );
1215
1216        if( job_info.failure ) {
1217          // Handle cleaning-up of job.
1218          remove_job( job, job_info );
1219          continue;
1220        }
1221
1222        boost::optional<JobState> new_state;
1223        while( (new_state = manage(job, job_info)) )
1224          job_info.state = *new_state;
1225
1226        if( job_info.starting_up )
1227          starting_up = true;
1228      }
1229
1230      while( !starting_up && !job_queue_->empty() &&
1231             job_map_.free_node_count() >= (job_queue_->front()->
    max_nodes == 1 ? 1 : 2) )
1232      {
1233        Job::ptr_t job = job_queue_->front();
1234        job_queue_->pop();
1235
```

```
1236        if( create_job(job) ) {
1237          starting_up = true;
1238          break;
1239        }
1240      }
1241    } while( process() );
1242
1243    se_running.stop();
1244    StatsEntry se_shutting_down( *this, "shutting_down", 100 );
1245
1246    // In case not all jobs finished successfully, log error message.
1247
1248    assert( job_queue_->empty() || jobs_done_ != job_queue_->total_jobs
     () );
1249
1250    if( jobs_done_ != job_queue_->total_jobs() ) {
1251      log_error( (boost::format("Failed to run all jobs from job_queue.\
     n"
1252                                "Only %u jobs out of %u completed
     successfully.")
1253                 % jobs_done_ % job_queue_->total_jobs()).str() );
1254    }
1255    else {
1256      log_info( (boost::format("Successfully completed all %u jobs from
     job_queue.")
1257                 % jobs_done_).str() );
1258    }
1259
1260    // Give all nodes the signal to shutdown gracefully. Should this
1261    // fail, handle_failure() will throw an exception, forcing the
1262    // system down.
1263
1264    for( int node = min_child(); node <= max_child(); ++node )
1265      shutdown( node );
1266
1267    while( process() );
1268
1269    // Output final statistics.
1270
1271    log_statistics();
1272  }
```

Listing B.32: src/manager_proc.hpp

```
1  #ifndef DICON_MANAGER_PROC_HPP_
2  #define DICON_MANAGER_PROC_HPP_
3
4  /*--- [Distribution] ------------------------------------------------
5   * This file is part of the Disease Control System DiCon.
6   *
7   * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8   * Designed and developed with the guidance of Nedialko B. Dimitrov
9   * and Lauren Ancel Meyers at the University of Texas at Austin.
10  *
11  * DiCon is free software: you can redistribute it and/or modify it
12  * under the terms of the GNU General Public License as published by
13  * the Free Software Foundation, either version 3 of the License, or
14  * (at your option) any later version.
15  *
16  * DiCon is distributed in the hope that it will be useful, but
17  * WITHOUT ANY WARRANTY; without even the implied warranty of
18  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19  * General Public License for more details.
20  *
21  * You should have received a copy of the GNU General Public License
22  * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23  *-----------------------------------------------------------------*/
24
25 /**
26  * @file
27  * @brief ProcNodeManager interface.
28  */
29 #include "manager.hpp"
30 #include "message.hpp"
31 #include <boost/mpi/request.hpp>
32 #include <deque>
33
34
35 namespace detail {
36   class QuestionDispatcher;
37 }
38
39 /**
40  * @brief Processing node manager interface.
41  *
42  * The ProcNodeManager class implements the interface to the node
43  * manager running on any processing node in the computing cluster. As
44  * such, it must be filled in with implementations of how the
45  * processing node manager processes each request sent to it by the
46  * main node.
47  *
48  * Each of those virtual methods to be implemented corresponds to the
49  * %message with the same name used for node intercommunication, as
50  * defined in the documentation of the message.hpp class, and the
51  * matching structures defined in the message::question and
52  * message::answer namespaces. See the documentation that is given
53  * there for details on each available %message and its parameters.
54  *
55  * Any of the processing functions (start_logging(), init_optimizer(),
56  * init_simulator(), init_combined(), get_policy(), simulate(),
57  * update(), step_combined(), dump_optimizer(), dump_policies(),
58  * shutdown()) is allowed to throw arbitrary exceptions. These will be
59  * caught inside the run() method which, instead of the expected
60  * answer %message holding the return value of the function called,
61  * will send an error %message back to the main node manager. In turn,
62  * the main node manager will deal with the situation as
63  * appropriate. See the documentation of the MainNodeManager class for
64  * details.
65  */
66 class ProcNodeManager
67   : public NodeManager
68 {
69   friend class detail::QuestionDispatcher;
70
71 protected:
72   /**
73    * @brief Create processing node manager.
74    *
75    * Constructor that creates a new node manager. This constructor is
76    * protected as the ProcNodeManager class is an abstract base class
```

```
77      * that does not allow direct instantiation.
78      *
79      * @param world MPI communicator.
80      */
81     ProcNodeManager( boost::mpi::communicator &world );
82
83   public:
84     virtual ~ProcNodeManager();
85
86   protected:
87     /// Process starting logging.
88     virtual void                    start_logging ( const boost::
    filesystem::path &logfile, LogLevel min_level                    ) =
    0;
89     /// Process initializing %optimizer.
90     virtual void                    init_optimizer( const boost::
    filesystem::path &optimizer_logfile
91                                                   , const boost::
    filesystem::path &simulator_logfile
92                                                   , const std::string
    &optimizer_library, const arguments_t &optimizer_arguments
93                                                   , const std::string
    &simulator_command, const arguments_t &simulator_arguments
94                                                   , const boost::
    optional<boost::filesystem::path> &optimizer_map_file
95                                                   , const boost::
    optional<boost::filesystem::path> &optimizer_lib_file            ) =
    0;
96     /// Process initializing simulator.
97     virtual void                    init_simulator( const boost::
    filesystem::path &simulator_logfile
98                                                   , const std::string
    &simulator_command, const arguments_t &simulator_arguments ) = 0;
99     /// Process initializing combined %optimizer/simulator.
100    virtual void                    init_combined ( const boost::
    filesystem::path &optimizer_logfile
101                                                   , const boost::
    filesystem::path &simulator_logfile
102                                                   , const std::string
    &optimizer_library, const arguments_t &optimizer_arguments
103                                                   , const std::string
    &simulator_command, const arguments_t &simulator_arguments
104                                                   , const boost::
    optional<boost::filesystem::path> &optimizer_map_file
105                                                   , const boost::
    optional<boost::filesystem::path> &optimizer_lib_file            ) =
    0;
106    /// Process getting next policy.
107    virtual boost::optional<policy_t> get_policy    (
                                                                     ) = 0;
108    /// Process simulating policy.
109    virtual double                  simulate      ( const policy_t &
    policy                                                           ) = 0;
110    /// Process updating reward.
111    virtual void                    update        ( const policy_t &
    policy, double reward                                            ) = 0;
112    /// Process taking step.
113    virtual bool                    step_combined (
                                                                     ) = 0;
114    /// Process dumping %optimizer state.
115    virtual void                    dump_optimizer( const boost::
    filesystem::path &file, DumpOptimizerMode mode                   ) =
    0;
116    /// Process dumping policies.
117    virtual void                    dump_policies ( const boost::
    filesystem::path &file, unsigned count, bool display             ) =
    0;
118    /// Process shutting down.
119    virtual bool                    shutdown      (
                                                                     ) = 0;
120
121  public:
122    /**
123     * @brief Run processing node manager.
124     *
125     * Run the processing node manager. This method waits for incoming
126     * requests from the main node manager and calls the corresponding
127     * processing function to each request. It stops doing so as soon as
128     * the shutdown message has been received and confirmed by the
129     * implementation.
130     *
131     * This method should only be called once during the lifetime of the
132     * ProcNodeManager object.
133     */
134    virtual void run();
135
136  private:
137    void process();
138
139  private:
140    bool shutdown_;
141    int current_tag_;
142    std::deque<boost::mpi::request> isend_queue_;
143  };
144
145  #endif //DICON_MANAGER_PROC_HPP_
```

Listing B.33: src/manager_proc.cpp

```
1   /*---- [Distribution] ------------------------------------------------
2    * This file is part of the Disease Control System DiCon.
3    *
4    * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
5    * Designed and developed with the guidance of Nedialko B. Dimitrov
6    * and Lauren Ancel Meyers at the University of Texas at Austin.
7    *
8    * DiCon is free software: you can redistribute it and/or modify it
9    * under the terms of the GNU General Public License as published by
10   * the Free Software Foundation, either version 3 of the License, or
11   * (at your option) any later version.
12   *
13   * DiCon is distributed in the hope that it will be useful, but
14   * WITHOUT ANY WARRANTY; without even the implied warranty of
15   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
16   * General Public License for more details.
17   *
18   * You should have received a copy of the GNU General Public License
19   * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
20   *--------------------------------------------------------------------*/
```

```
21
22 #include "manager_proc.hpp"
23 #include "log.hpp"
24 #include <boost/format.hpp>
25 #include <boost/mpi/communicator.hpp>
26
27
28 namespace detail {
29
30   static const size_t isend_queue_size = 10;
31
32 }
33
34
35
36 namespace detail {
37
38   class QuestionDispatcher
39     : public boost::static_visitor<message::Answer>
40   {
41   public:
42     QuestionDispatcher( ProcNodeManager &manager );
43
44   public:
45     message::Answer operator()( const message::question::StartLogging &data ) const;
46     message::Answer operator()( const message::question::InitOptimizer &data ) const;
47     message::Answer operator()( const message::question::InitSimulator &data ) const;
48     message::Answer operator()( const message::question::InitCombined &data ) const;
49     message::Answer operator()( const message::question::GetPolicy &data ) const;
50     message::Answer operator()( const message::question::Simulate &data ) const;
51     message::Answer operator()( const message::question::Update &data ) const;
52     message::Answer operator()( const message::question::StepCombined &data ) const;
53     message::Answer operator()( const message::question::DumpOptimizer &data ) const;
54     message::Answer operator()( const message::question::DumpPolicies &data ) const;
55     message::Answer operator()( const message::question::Shutdown &data ) const;
56
57   private:
58     ProcNodeManager &manager_;
59   };
60
61 }
62
63
64
65 namespace detail {
66
67   template< typename A >
68   static
69   message::Answer
70   make_answer( const A &answer ) {
71     message::Answer res;
72     res.data = answer;
73     return res;
74   }
75
76
77   static
78   message::Answer
79   make_failure( const std::string &what ) {
80     message::answer::Failure failure;
81     failure.what = what;
82
83     return make_answer( failure );
84   }
85
86
87   static
88   inline
89   void
90   log_debug( const std::string &what ) throw() {
91     try {
92       ::log_debug( what );
93     }
94     catch( ... ) {
95       // Ignore.
96     }
97   }
98
99
100  class LogDebug
101    : boost::noncopyable
102  {
103  public:
104    LogDebug( const std::string &what )
105      : what_(what)
106    {
107      log_debug(              what_ + "()" );
108    }
109
110    ~LogDebug() {
111      log_debug( "finish_" + what_ + "()" );
112    }
113
114  private:
115    std::string what_;
116  };
117
118
119  QuestionDispatcher::QuestionDispatcher( ProcNodeManager &manager )
120    : manager_(manager)
121  {
122  }
123
124
125  message::Answer
126  QuestionDispatcher::operator()( const message::question::StartLogging &data ) const {
127    ProcNodeManager::StatsEntry se( manager_, "start_logging()", 10 );
128    LogDebug log_debug( "start_logging" );
129
130    message::answer::StartLogging answer;
```

```
131
132     manager_.start_logging( data.logfile, data.min_level );
133
134     return make_answer( answer );
135   }
136
137
138   message::Answer
139   QuestionDispatcher::operator()( const message::question::InitOptimizer &data ) const {
140     ProcNodeManager::StatsEntry se( manager_, "init_optimizer()", 20 );
141     LogDebug log_debug( "init_optimizer" );
142
143     message::answer::InitOptimizer answer;
144
145     manager_.init_optimizer( data.optimizer_logfile, data.simulator_logfile,
146                              data.optimizer_library, data.optimizer_arguments,
147                              data.simulator_command, data.simulator_arguments,
148                              data.optimizer_map_file, data.optimizer_lib_file );
149
150     return make_answer( answer );
151   }
152
153
154   message::Answer
155   QuestionDispatcher::operator()( const message::question::InitSimulator &data ) const {
156     ProcNodeManager::StatsEntry se( manager_, "init_simulator()", 30 );
157     LogDebug log_debug( "init_simulator" );
158
159     message::answer::InitSimulator answer;
160
161     manager_.init_simulator( data.simulator_logfile,
162                              data.simulator_command, data.simulator_arguments );
163
164     return make_answer( answer );
165   }
166
167
168   message::Answer
169   QuestionDispatcher::operator()( const message::question::InitCombined &data ) const {
170     ProcNodeManager::StatsEntry se( manager_, "init_combined()", 40 );
171     LogDebug log_debug( "init_combined" );
172
173     message::answer::InitCombined answer;
174
175     manager_.init_combined( data.optimizer_logfile, data.simulator_logfile,
176                             data.optimizer_library, data.optimizer_arguments,
177                             data.simulator_command, data.simulator_arguments,
178                             data.optimizer_map_file, data.optimizer_lib_file );
179
180     return make_answer( answer );
181   }
182
183
184   message::Answer
185   QuestionDispatcher::operator()( const message::question::GetPolicy &data ) const {
186     ProcNodeManager::StatsEntry se( manager_, "get_policy()", 50 );
187     LogDebug log_debug( "get_policy" );
188
189     message::answer::GetPolicy answer;
190
191     answer.policy = manager_.get_policy();
192
193     return make_answer( answer );
194   }
195
196
197   message::Answer
198   QuestionDispatcher::operator()( const message::question::Simulate &data ) const {
199     ProcNodeManager::StatsEntry se( manager_, "simulate()", 60 );
200     LogDebug log_debug( "simulate" );
201
202     message::answer::Simulate answer;
203
204     answer.reward = manager_.simulate( data.policy );
205
206     return make_answer( answer );
207   }
208
209
210   message::Answer
211   QuestionDispatcher::operator()( const message::question::Update &data ) const {
212     ProcNodeManager::StatsEntry se( manager_, "update()", 70 );
213     LogDebug log_debug( "update" );
214
215     message::answer::Update answer;
216
217     manager_.update( data.policy, data.reward );
218
219     return make_answer( answer );
220   }
221
222
223   message::Answer
224   QuestionDispatcher::operator()( const message::question::StepCombined &data ) const {
225     ProcNodeManager::StatsEntry se( manager_, "step_combined()", 80 );
226     LogDebug log_debug( "step_combined" );
227
228     message::answer::StepCombined answer;
229
230     answer.got_policy = manager_.step_combined();
231
232     return make_answer( answer );
233   }
```

```
234
235
236   message::Answer
237   QuestionDispatcher::operator()( const message::question::DumpOptimizer &data ) const {
238     ProcNodeManager::StatsEntry se( manager_, "dump_optimizer()", 90 );
239     LogDebug log_debug( "dump_optimizer" );
240
241     message::answer::DumpOptimizer answer;
242
243     manager_.dump_optimizer( data.file, data.mode );
244
245     return make_answer( answer );
246   }
247
248
249   message::Answer
250   QuestionDispatcher::operator()( const message::question::DumpPolicies &data ) const {
251     ProcNodeManager::StatsEntry se( manager_, "dump_policies()", 100 );
252     LogDebug log_debug( "dump_policies" );
253
254     message::answer::DumpPolicies answer;
255
256     manager_.dump_policies( data.file, data.count, data.display );
257
258     return make_answer( answer );
259   }
260
261
262   message::Answer
263   QuestionDispatcher::operator()( const message::question::Shutdown &data ) const {
264     ProcNodeManager::StatsEntry se( manager_, "shutdown()", 110 );
265     LogDebug log_debug( "shutdown" );
266
267     message::answer::Shutdown answer;
268
269     answer.done = manager_.shutdown();
270
271     if( answer.done )
272       manager_.shutdown_ = true;
273
274     return make_answer( answer );
275   }
276
277 }
278
279
280
281 ProcNodeManager::ProcNodeManager( boost::mpi::communicator &world )
282   : NodeManager(world), shutdown_(false), current_tag_(min_tag())
283 {
284   StatsEntry se( *this, "starting_up", 0 );
285 }
286
287
288 ProcNodeManager::~ProcNodeManager() {
289   assert( isend_queue_.empty() );
290 }
291
292
293 void
294 ProcNodeManager::process() {
295   StatsEntry se( *this, "processing", 0, "(idle)", 0 );
296
297   assert( isend_queue_.size() <= detail::isend_queue_size );
298   if( isend_queue_.size() == detail::isend_queue_size ) {
299     assert( !isend_queue_.empty() );
300     isend_queue_.front().wait();
301     isend_queue_.pop_front();
302   }
303
304   const unsigned tag = next_tag(current_tag_);
305
306   message::Question question;
307
308   world().recv( 0, tag, question );
309
310   boost::optional<message::Answer> answer;
311
312   try {
313     answer = boost::apply_visitor( detail::QuestionDispatcher(*this), question.data );
314   }
315 #if BOOST_VERSION >= 103900
316   catch( ... ) {
317     const std::string &info = boost::current_exception_diagnostic_information();
318     log_error( (boost::format("Failed to process request.\n%s") % info).str() );
319     answer = detail::make_failure( info );
320   }
321 #else
322   catch( boost::exception &e ) {
323     const std::string &info = boost::diagnostic_information( e );
324     log_error( (boost::format("Failed to process request.\n%s") % info).str() );
325     answer = detail::make_failure( info );
326   }
327   catch( ... ) {
328     const std::string &info = "<unknown exception>\n";
329     log_error( (boost::format("Failed to process request.\n%s") % info).str() );
330     answer = detail::make_failure( info );
331   }
332 #endif
333
334   isend_queue_.push_back( world().isend(0, tag, answer.get()) );
335 }
336
337
338 void
339 ProcNodeManager::run() {
340   StatsEntry se_running( *this, "running", 50 );
341
342   while( !shutdown_ ) {
343     process();
344   }
345
```

```
346     se_running.stop();
347     StatsEntry se_shutting_down( *this, "shutting_down", 0 );
348
349     while( !isend_queue_.empty() ) {
350       isend_queue_.front().wait();
351       isend_queue_.pop_front();
352     }
353
354     // Output final statistics.
355
356     log_statistics();
357   }
```

Listing B.34: src/manager_proc_impl.hpp

```
1   #ifndef DICON_MANAGER_PROC_IMPL_HPP_
2   #define DICON_MANAGER_PROC_IMPL_HPP_
3
4   /*--- [Distribution] ------------------------------------------------------
5    * This file is part of the Disease Control System DiCon.
6    *
7    * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
8    * Designed and developed with the guidance of Nedialko B. Dimitrov
9    * and Lauren Ancel Meyers at the University of Texas at Austin.
10   *
11   * DiCon is free software: you can redistribute it and/or modify it
12   * under the terms of the GNU General Public License as published by
13   * the Free Software Foundation, either version 3 of the License, or
14   * (at your option) any later version.
15   *
16   * DiCon is distributed in the hope that it will be useful, but
17   * WITHOUT ANY WARRANTY; without even the implied warranty of
18   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
19   * General Public License for more details.
20   *
21   * You should have received a copy of the GNU General Public License
22   * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
23   *-----------------------------------------------------------------------*/
24
25  /**
26   * @file
27   * @brief ProcNodeManagerImpl class.
28   */
29  #include "manager_proc.hpp"
30  #include "optimizer.hpp"
31  #include "simulator.hpp"
32
33
34  class LogFile;
35
36  /**
37   * @brief Processing node manager implementation.
38   *
39   * The ProcNodeManagerImpl class implements the node manager running
40   * on any processing node inside the computing cluster. Specifically,
41   * it handles all the requests sent to it by the main node manager in
42   * order to achieve the required functionality.
43   *
44   * As defined in the NodeManager interface, the processing node
45   * manager is run by calling the run() method.
46   */
47  class ProcNodeManagerImpl
48    : public ProcNodeManager
49  {
50    friend class NodeManager;
51
52  protected:
53    /**
54     * @brief Create processing node manager.
55     *
56     * Constructor that creates a new processing node manager. This
57     * constructor is protected since objects of the ProcNodeManagerImpl
58     * class are only to be costructed through the static @link
59     * NodeManager::create() create()@endlink method of the NodeManager
60     * class.
61     *
62     * @param world MPI communicator.
63     */
64    ProcNodeManagerImpl( boost::mpi::communicator &world );
65
66  protected:
67    virtual void                    start_logging ( const boost::
    (filesystem::path &logfile, LogLevel min_level                    );
68    virtual void                    init_optimizer( const boost::
    (filesystem::path &optimizer_logfile
69                                                  , const boost::
    (filesystem::path &simulator_logfile
70                                                  , const std::string
    (&optimizer_library, const arguments_t &optimizer_arguments
71                                                  , const std::string
    (&simulator_command, const arguments_t &simulator_arguments
72                                                  , const boost::
    (optional<boost::filesystem::path> &optimizer_map_file
73                                                  , const boost::
    (optional<boost::filesystem::path> &optimizer_lib_file            );
74    virtual void                    init_simulator( const boost::
    (filesystem::path &simulator_logfile
75                                                  , const std::string
    (&simulator_command, const arguments_t &simulator_arguments );
76    virtual void                    init_combined ( const boost::
    (filesystem::path &optimizer_logfile
77                                                  , const boost::
    (filesystem::path &simulator_logfile
78                                                  , const std::string
    (&optimizer_library, const arguments_t &optimizer_arguments
79                                                  , const std::string
    (&simulator_command, const arguments_t &simulator_arguments
80                                                  , const boost::
    (optional<boost::filesystem::path> &optimizer_map_file
81                                                  , const boost::
    (optional<boost::filesystem::path> &optimizer_lib_file            );
82    virtual boost::optional<policy_t> get_policy    (                 )
    (                                                                 );
83    virtual double                  simulate      ( const policy_t &)
    (policy                                                           );
84    virtual void                    update        ( const policy_t &)
    (policy, double reward                                            );
85    virtual bool                    step_combined (                   )
    (                                                                 );
```

```
86    virtual void                        dump_optimizer( const boost::
      filesystem::path &file, DumpOptimizerMode mode                    );
87    virtual void                        dump_policies ( const boost::
      filesystem::path &file, unsigned count, bool display              );
88    virtual bool                        shutdown      (                   
                                                                        );
89
90  private:
91    void reset_all( bool logfile = false );
92
93  private:
94    boost::shared_ptr<LogFile> logfile_;
95    boost::scoped_ptr<Optimizer> optimizer_;
96    boost::scoped_ptr<Simulator> simulator_;
97    boost::scoped_ptr<Combined > combined_;
98  };
99
100 #endif //DICON_MANAGER_PROC_IMPL_HPP_
```

Listing B.35: src/manager_proc_impl.cpp

```
1   /*--- [Distribution] ------------------------------------------------------
2    * This file is part of the Disease Control System DiCon.
3    *
4    * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
5    * Designed and developed with the guidance of Nedialko B. Dimitrov
6    * and Lauren Ancel Meyers at the University of Texas at Austin.
7    *
8    * DiCon is free software: you can redistribute it and/or modify it
9    * under the terms of the GNU General Public License as published by
10   * the Free Software Foundation, either version 3 of the License, or
11   * (at your option) any later version.
12   *
13   * DiCon is distributed in the hope that it will be useful, but
14   * WITHOUT ANY WARRANTY; without even the implied warranty of
15   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
16   * General Public License for more details.
17   *
18   * You should have received a copy of the GNU General Public License
19   * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
20   *-----------------------------------------------------------------------*/
21
22  #include "manager_proc_impl.hpp"
23  #include "error.hpp"
24  #include "log.hpp"
25
26
27  ProcNodeManagerImpl::ProcNodeManagerImpl( boost::mpi::communicator &
    world )
28    : ProcNodeManager(world)
29  {
30  }
31
32
33  void
34  ProcNodeManagerImpl::start_logging( const boost::filesystem::path &
    logfile, LogLevel min_level ) {
35    logfile_.reset( new LogFile(logfile) );
36    log_register_sink( logfile_, min_level );
37  }
38
39
40  void
41  ProcNodeManagerImpl::init_optimizer( const boost::filesystem::path &
    optimizer_logfile
42                                       , const boost::filesystem::path &
    simulator_logfile
43                                       , const std::string &
    optimizer_library, const arguments_t &optimizer_arguments
44                                       , const std::string &
    simulator_command, const arguments_t &simulator_arguments
45                                       , const boost::optional<boost::
    filesystem::path> &optimizer_map_file
46                                       , const boost::optional<boost::
    filesystem::path> &optimizer_lib_file
47                                       )
48  {
49    reset_all();
50    optimizer_.reset( new Optimizer(optimizer_logfile, simulator_logfile
    ,
51                                    optimizer_library, 
    optimizer_arguments,
52                                    simulator_command, 
    simulator_arguments,
53                                    optimizer_map_file, 
    optimizer_lib_file) );
54  }
55
56
57  void
58  ProcNodeManagerImpl::init_simulator( const boost::filesystem::path &
    simulator_logfile
59                                       , const std::string &
    simulator_command, const arguments_t &simulator_arguments
60                                       )
61  {
62    reset_all();
63    simulator_.reset( new Simulator(simulator_logfile,
64                                    simulator_command, 
    simulator_arguments) );
65  }
66
67
68  void
69  ProcNodeManagerImpl::init_combined( const boost::filesystem::path &
    optimizer_logfile
70                                      , const boost::filesystem::path &
    simulator_logfile
71                                      , const std::string &
    optimizer_library, const arguments_t &optimizer_arguments
72                                      , const std::string &
    simulator_command, const arguments_t &simulator_arguments
73                                      , const boost::optional<boost::
    filesystem::path> &optimizer_map_file
74                                      , const boost::optional<boost::
    filesystem::path> &optimizer_lib_file
75                                      )
76  {
77    reset_all();
78    combined_.reset( new Combined(optimizer_logfile, simulator_logfile,
```

```
                                optimizer_library, optimizer_arguments,
                                simulator_command, simulator_arguments,
                                optimizer_map_file, optimizer_lib_file) );
}


boost::optional<policy_t>
ProcNodeManagerImpl::get_policy() {
  if( !optimizer_ )
    BOOST_THROW_EXCEPTION( AssertionError() );

  return optimizer_->get_policy();
}


double
ProcNodeManagerImpl::simulate( const policy_t &policy ) {
  if( !simulator_ )
    BOOST_THROW_EXCEPTION( AssertionError() );

  return simulator_->simulate( policy );
}


void
ProcNodeManagerImpl::update( const policy_t &policy, double reward ) {
  if( !optimizer_ )
    BOOST_THROW_EXCEPTION( AssertionError() );

  optimizer_->update( policy, reward );
}


bool
ProcNodeManagerImpl::step_combined() {
  if( !combined_ )
    BOOST_THROW_EXCEPTION( AssertionError() );

  return combined_->step();
}



namespace detail {

  template< typename T >
  static
  bool
  dump_optimizer( const boost::scoped_ptr<T> &optimizer,
                  const boost::filesystem::path &file, DumpOptimizerMode mode )
  {
    if( optimizer ) {
      switch( mode ) {
      case DUMP_OPTIMIZER_MAP: optimizer->dump_optimizer_map( file ); break;
      case DUMP_OPTIMIZER_LIB: optimizer->dump_optimizer_lib( file ); break;
      }
    }

    return bool(optimizer);
  }


  template< typename T >
  static
  bool
  dump_policies( const boost::scoped_ptr<T> &optimizer,
                 const boost::filesystem::path &file, unsigned count, bool display )
  {
    if( optimizer )
      optimizer->dump_policies( file, count, display );

    return bool(optimizer);
  }

}



void
ProcNodeManagerImpl::dump_optimizer( const boost::filesystem::path &file, DumpOptimizerMode mode ) {
  if(  !detail::dump_optimizer(optimizer_, file, mode)
    && !detail::dump_optimizer(combined_, file, mode) )
  {
    BOOST_THROW_EXCEPTION( AssertionError() );
  }
}


void
ProcNodeManagerImpl::dump_policies( const boost::filesystem::path &file, unsigned count, bool display ) {
  if(  !detail::dump_policies(optimizer_, file, count, display)
    && !detail::dump_policies(combined_, file, count, display) )
  {
    BOOST_THROW_EXCEPTION( AssertionError() );
  }
}


bool
ProcNodeManagerImpl::shutdown() {
  log_statistics();

  reset_all( true );

  return true;
}


void
ProcNodeManagerImpl::reset_all( bool logfile ) {
  if( logfile )
    logfile_.reset();
```

```
192     optimizer_.reset();
193     simulator_.reset();
194     combined_.reset();
195 }
```

Listing B.36: src/message.hpp

```
1  #ifndef DICON_MESSAGE_HPP_
2  #define DICON_MESSAGE_HPP_
3
4  /*---- [Distribution] ------------------------------------------------
5   * This file is part of the Disease Control System DiCon.
6   *
7   * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
8   * Designed and developed with the guidance of Nedialko B. Dimitrov
9   * and Lauren Ancel Meyers at the University of Texas at Austin.
10  *
11  * DiCon is free software: you can redistribute it and/or modify it
12  * under the terms of the GNU General Public License as published by
13  * the Free Software Foundation, either version 3 of the License, or
14  * (at your option) any later version.
15  *
16  * DiCon is distributed in the hope that it will be useful, but
17  * WITHOUT ANY WARRANTY; without even the implied warranty of
18  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
19  * General Public License for more details.
20  *
21  * You should have received a copy of the GNU General Public License
22  * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
23  *--------------------------------------------------------------------*/
24
25 /**
26  * @file
27  * @brief Node intercommunication.
28  *
29  * The message.hpp file provides the structures necessary for node
30  * intercommunication between the main node (MainNodeManager
31  * interface, MainNodeManagerImpl class) and the processing nodes
32  * (ProcNodeManager interface, ProcNodeManagerImpl class). The
33  * following messages are defined.
34  *
35  * - @c start_logging(): Start logging to the given file with the
36  *   given minimum log level.
37  * - @c init_optimizer(): Initialize %optimizer instance with the
38  *   given %optimizer library and arguments, and the given simulator
39  *   command and arguments, using the given logfiles for the
40  *   %optimizer and simulator; optionally restoring an %optimizer
41  *   state from the given map and lib dumpfiles. This will shutdown
42  *   any other %optimizer, simulator, or combined %optimizer/simulator
43  *   running at the target node.
44  * - @c init_simulator(): Initialize simulator instance with the given
45  *   simulator command and arguments, using the given logfile for the
46  *   simulator. This will shutdown any other %optimizer, simulator, or
47  *   combined %optimizer/simulator running at the target node.
48  * - @c init_combined(): Initialize combined %optimizer/simulator
49  *   instance with the given %optimizer library and arguments, and the
50  *   given simulator command and arguments, using the given logfiles
51  *   for the %optimizer and simulator; optionally restoring an
52  *   %optimizer state from the given map and lib dumpfiles. This will
53  *   shutdown any other %optimizer, simulator, or combined
54  *   %optimizer/simulator running at the target node.
55  * - @c get_policy(): Get next policy from %optimizer, or a special
56  *   value indicating that no policy is available until @c update() is
57  *   called at least once. This call is only valid after an %optimizer
58  *   (@c init_optimizer()) has been initialized on the target node.
59  * - @c simulate(): Simulate the given policy. This call is only valid
60  *   after a simulator (@c init_simulator()) has been initialized on
61  *   the target node.
62  * - @c update(): Update the reward for the given policy which must
63  *   have been returned by the target node on a previous call to @c
64  *   get_policy(). This call is only valid after an %optimizer (@c
65  *   init_optimizer()) has been initialized on the target node.
66  * - @c step_combined(): Do another step, i.e., get policy, simulate,
67  *   and update. This call is only valid after a combined
68  *   %optimizer/simulator (@c init_combined()) has been initialized on
69  *   the target node.
70  * - @c dump_optimizer(): Dump current %optimizer state to the given
71  *   file (either map or lib state). This call is only valid after an
72  *   %optimizer (@c init_optimizer()) or combined %optimizer/simulator
73  *   (@c init_combined()) has been initialized on the target node.
74  * - @c dump_policies(): Dump list of best policies to the given file
75  *   (either binary or textual). This call is only valid after an
76  *   %optimizer (@c init_optimizer()) or combined %optimizer/simulator
77  *   (@c init_combined()) has been initialized on the target node.
78  * - @c shutdown(): Shutdown the node. After this %message has been
79  *   confirmed, no more messages are processed at the target node.
80  *   Also, logging to a file (@c start_logging()) will be stopped.
81  *
82  * The @c dump_optimizer() %message defines two dump modes
83  * (DumpOptimizerMode enum), used for dumping different aspects of the
84  * %optimizer state:
85  *
86  * - DUMP_OPTIMIZER_MAP: Dump of the one-to-one correspondence between
87  *   policies as returned by the simulator's @c children() method, and
88  *   the @c int values handed over to the optimization algorithm.
89  * - DUMP_OPTIMIZER_LIB: Dump of the %optimizer's own internal
90  *   state. This data belongs to the optimization algorithm alone and
91  *   can have an arbitrary format.
92  *
93  * To each %message defined above there is a corresponding pair of
94  * structures in the message::question and message::answer namespaces.
95  * The corresponding methods in the MainNodeManager interface send
96  * serialized question objects that the ProcNodeManager interface
97  * receives. After calling the corresponding method in the
98  * ProcNodeManagerImpl class, an answer object is sent back and will
99  * be handled by calling a corresponding <code>finish_</code> method
100 * in the MainNodeManagerImpl class.
101 *
102 * @note The target node will never overwrite a logfile that already
103 *   exists. If the filename of a logfile given in any of the
104 *   messages @c start_logging(), @c init_optimizer(), @c
105 *   init_simulator(), or @c init_combined() indicates an existing
106 *   file, the target node will choose another name by the means of
107 *   File::unique(). This behavior is used exclusively for logfiles
108 *   and is @e not applied to dump files, as in @c dump_optimizer(),
109 *   or @c dump_policies(): with these messages, the target node will
110 *   @e always use the given filename, overwriting any files as
111 *   necessary.
112 */
113 #include "types.hpp"
```

```
#include <boost/filesystem/path.hpp>
#include <boost/serialization/optional.hpp>
#include <boost/serialization/string.hpp>
#include <boost/serialization/variant.hpp>
#include <boost/serialization/vector.hpp>


/// Dump mode used for @c dump_optimizer() %message.
enum DumpOptimizerMode
{ DUMP_OPTIMIZER_MAP
, DUMP_OPTIMIZER_LIB
};

/// Node intercommunication messages.
namespace message {
  /// Messages from main node to processing node.
  namespace question {

    /**
     * Message @c start_logging().
     * @see message.hpp.
     */
    struct StartLogging {
      /// Logfile to start logging to.
      boost::filesystem::path logfile;
      /// Minimum log level for file.
      LogLevel min_level;
    };

    /**
     * Message @c init_optimizer().
     * @see message.hpp.
     */
    struct InitOptimizer {
      /// Logfile to use for logging %optimizer error output.
      boost::filesystem::path optimizer_logfile;
      /// Filename of the %optimizer library.
      std::string optimizer_library;
      /// %Optimizer library arguments.
      arguments_t optimizer_arguments;
      /// %Optimizer map dumpfile to restore.
      boost::optional<boost::filesystem::path> optimizer_map_file;
      /// %Optimizer lib dumpfile to restore.
      boost::optional<boost::filesystem::path> optimizer_lib_file;

      /// Logfile to use for logging simulator error output.
      boost::filesystem::path simulator_logfile;
      /// Filename of the simulator command.
      std::string simulator_command;
      /// %Simulator command arguments.
      arguments_t simulator_arguments;
    };

    /**
     * Message @c init_simulator().
     * @see message.hpp.
     */
    struct InitSimulator {
      /// Logfile to use for logging simulator error output.
      boost::filesystem::path simulator_logfile;
      /// Filename of the simulator command.
      std::string simulator_command;
      /// %Simulator command arguments.
      arguments_t simulator_arguments;
    };

    /**
     * Message @c init_combined().
     * @see message.hpp.
     */
    struct InitCombined {
      /// Logfile to use for logging %optimizer error output.
      boost::filesystem::path optimizer_logfile;
      /// Filename of the %optimizer library.
      std::string optimizer_library;
      /// %Optimizer library arguments.
      arguments_t optimizer_arguments;
      /// %Optimizer map dumpfile to restore.
      boost::optional<boost::filesystem::path> optimizer_map_file;
      /// %Optimizer lib dumpfile to restore.
      boost::optional<boost::filesystem::path> optimizer_lib_file;

      /// Logfile to use for logging simulator error output.
      boost::filesystem::path simulator_logfile;
      /// Filename of the simulator command.
      std::string simulator_command;
      /// %Simulator command arguments.
      arguments_t simulator_arguments;
    };

    /**
     * Message @c get_policy().
     * @see message.hpp.
     */
    struct GetPolicy {
    };

    /**
     * Message @c simulate().
     * @see message.hpp.
     */
    struct Simulate {
      /// Policy to simulate.
      policy_t policy;
    };

    /**
     * Message @c update().
     * @see message.hpp.
     */
    struct Update {
      /// Simulated policy.
      policy_t policy;
      /// Policy's reward.
      double reward;
    };

    /**
     * Message @c step_combined().
     * @see message.hpp.
     */
    struct StepCombined {
```

```
    };

    /**
     * Message @c dump_optimizer().
     * @see message.hpp.
     */
    struct DumpOptimizer {
      /// Filename of %optimizer dump file.
      boost::filesystem::path file;
      /// Dump mode to use for file.
      DumpOptimizerMode mode;
    };

    /**
     * Message @c dump_policies().
     * @see message.hpp.
     */
    struct DumpPolicies {
      /// Filename of policy list file.
      boost::filesystem::path file;
      /// Maximum number of policies.
      unsigned count;
      /// Use textual display for policies?
      bool display;
    };

    /**
     * Message @c shutdown().
     * @see message.hpp.
     */
    struct Shutdown {
    };

  }
}


namespace message {
  /// Messages from processing node to main node.
  namespace answer {

    /**
     * Message @c start_logging().
     * @see message.hpp.
     */
    struct StartLogging {
    };

    /**
     * Message @c init_optimizer().
     * @see message.hpp.
     */
    struct InitOptimizer {
    };

    /**
     * Message @c init_simulator().
     * @see message.hpp.
     */
    struct InitSimulator {
    };

    /**
     * Message @c init_combined().
     * @see message.hpp.
     */
    struct InitCombined {
    };

    /**
     * Message @c get_policy().
     * @see message.hpp.
     */
    struct GetPolicy {
      /// Policy.
      boost::optional<policy_t> policy;
    };

    /**
     * Message @c simulate().
     * @see message.hpp.
     */
    struct Simulate {
      /// Reward.
      double reward;
    };

    /**
     * Message @c update().
     * @see message.hpp.
     */
    struct Update {
    };

    /**
     * Message @c step_combined().
     * @see message.hpp.
     */
    struct StepCombined {
      /// Policy was available?
      bool got_policy;
    };

    /**
     * Message @c dump_optimizer().
     * @see message.hpp.
     */
    struct DumpOptimizer {
    };

    /**
     * Message @c dump_policies().
     * @see message.hpp.
     */
    struct DumpPolicies {
    };

    /**
     * Message @c shutdown().
     * @see message.hpp.
     */
    struct Shutdown {
```

```
358       /// Node was shutdown?
359       bool done;
360     };
361
362     /**
363      * %Failure.
364      * @see message.hpp.
365      */
366     struct Failure {
367       /// Description of error.
368       std::string what;
369     };
370
371   }
372 }
373
374
375 namespace message {
376
377   // Order of different %message types has to be the same for Question
378   // and Answer: we have  to use boost::variant::which() on those data
379   // attributes.
380
381   // Main node -> processing node.
382   struct Question {
383     boost::variant< question::StartLogging
384                   , question::InitOptimizer
385                   , question::InitSimulator
386                   , question::InitCombined
387                   , question::GetPolicy
388                   , question::Simulate
389                   , question::Update
390                   , question::StepCombined
391                   , question::DumpOptimizer
392                   , question::DumpPolicies
393                   , question::Shutdown
394                   > data;
395   };
396
397   // Processing node -> main node.
398   struct Answer {
399     boost::variant< answer::StartLogging
400                   , answer::InitOptimizer
401                   , answer::InitSimulator
402                   , answer::InitCombined
403                   , answer::GetPolicy
404                   , answer::Simulate
405                   , answer::Update
406                   , answer::StepCombined
407                   , answer::DumpOptimizer
408                   , answer::DumpPolicies
409                   , answer::Shutdown
410                   , answer::Failure
411                   > data;
412   };
413
414 }
415
416
417 BOOST_SERIALIZATION_SPLIT_FREE( boost::filesystem::path )
418
```

```
419 namespace boost {
420   namespace serialization {
421
422     template< class Archive >
423     void save( Archive &ar, const boost::filesystem::path &path, const
    unsigned int version );
424
425     template< class Archive >
426     void load( Archive &ar, boost::filesystem::path &path, const
    unsigned int version );
427
428   }
429 }
430
431
432 namespace boost {
433   namespace serialization {
434
435     template< class Archive >
436     void serialize( Archive &ar, message::Question &question, const
    unsigned int version );
437
438     template< class Archive >
439     void serialize( Archive &ar, message::question::StartLogging &data
    , const unsigned int version );
440
441     template< class Archive >
442     void serialize( Archive &ar, message::question::InitOptimizer &
    data, const unsigned int version );
443
444     template< class Archive >
445     void serialize( Archive &ar, message::question::InitSimulator &
    data, const unsigned int version );
446
447     template< class Archive >
448     void serialize( Archive &ar, message::question::InitCombined &data
    , const unsigned int version );
449
450     template< class Archive >
451     void serialize( Archive &ar, message::question::GetPolicy &data,
    const unsigned int version );
452
453     template< class Archive >
454     void serialize( Archive &ar, message::question::Simulate &data,
    const unsigned int version );
455
456     template< class Archive >
457     void serialize( Archive &ar, message::question::Update &data,
    const unsigned int version );
458
459     template< class Archive >
460     void serialize( Archive &ar, message::question::StepCombined &data
    , const unsigned int version );
461
462     template< class Archive >
463     void serialize( Archive &ar, message::question::DumpOptimizer &
    data, const unsigned int version );
464
465     template< class Archive >
466     void serialize( Archive &ar, message::question::DumpPolicies &data
    , const unsigned int version );
```

```
467
468     template< class Archive >
469     void serialize( Archive &ar, message::question::Shutdown &data,
    const unsigned int version );
470
471   }
472 }
473
474
475 namespace boost {
476   namespace serialization {
477
478     template< class Archive >
479     void serialize( Archive &ar, message::Answer &answer, const
    unsigned int version );
480
481     template< class Archive >
482     void serialize( Archive &ar, message::answer::StartLogging &data,
    const unsigned int version );
483
484     template< class Archive >
485     void serialize( Archive &ar, message::answer::InitOptimizer &data,
     const unsigned int version );
486
487     template< class Archive >
488     void serialize( Archive &ar, message::answer::InitSimulator &data,
     const unsigned int version );
489
490     template< class Archive >
491     void serialize( Archive &ar, message::answer::InitCombined &data,
    const unsigned int version );
492
493     template< class Archive >
494     void serialize( Archive &ar, message::answer::GetPolicy &data,
    const unsigned int version );
495
496     template< class Archive >
497     void serialize( Archive &ar, message::answer::Simulate &data,
    const unsigned int version );
498
499     template< class Archive >
500     void serialize( Archive &ar, message::answer::Update &data, const
    unsigned int version );
501
502     template< class Archive >
503     void serialize( Archive &ar, message::answer::StepCombined &data,
    const unsigned int version );
504
505     template< class Archive >
506     void serialize( Archive &ar, message::answer::DumpOptimizer &data,
     const unsigned int version );
507
508     template< class Archive >
509     void serialize( Archive &ar, message::answer::DumpPolicies &data,
    const unsigned int version );
510
511     template< class Archive >
512     void serialize( Archive &ar, message::answer::Shutdown &data,
    const unsigned int version );
513
514     template< class Archive >
515     void serialize( Archive &ar, message::answer::Failure &data, const
    unsigned int version );
516
517   }
518 }
519
520
521 #include "message.ipp"
522
523 #endif //DICON_MESSAGE_HPP_
```

Listing B.37: src/message.ipp

```
1  // -*- c++ -*-
2
3  /*---- [Distribution] ------------------------------------------------
4   * This file is part of the Disease Control System DiCon.
5   *
6   * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
7   * Designed and developed with the guidance of Nedialko B. Dimitrov
8   * and Lauren Ancel Meyers at the University of Texas at Austin.
9   *
10  * DiCon is free software: you can redistribute it and/or modify it
11  * under the terms of the GNU General Public License as published by
12  * the Free Software Foundation, either version 3 of the License, or
13  * (at your option) any later version.
14  *
15  * DiCon is distributed in the hope that it will be useful, but
16  * WITHOUT ANY WARRANTY; without even the implied warranty of
17  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
18  * General Public License for more details.
19  *
20  * You should have received a copy of the GNU General Public License
21  * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
22  *--------------------------------------------------------------------*/
23
24 namespace boost {
25   namespace serialization {
26
27     template< class Archive >
28     inline
29     void
30     save( Archive &ar, const boost::filesystem::path &path, const
    unsigned int version ) {
31       std::string str = path.string();
32       ar << str;
33     }
34
35     template< class Archive >
36     inline
37     void
38     load( Archive &ar, boost::filesystem::path &path, const unsigned
    int version ) {
39       std::string str;
40       ar >> str;
41       path = str;
42     }
43
44   }
```

```
}


namespace boost {
  namespace serialization {

    template< class Archive >
    inline
    void
    serialize( Archive &ar, message::question::StartLogging &data, const unsigned int version ) {
      ar & data.logfile;
      ar & data.min_level;
    }

    template< class Archive >
    inline
    void
    serialize( Archive &ar, message::question::InitOptimizer &data, const unsigned int version ) {
      ar & data.optimizer_logfile;
      ar & data.optimizer_library;
      ar & data.optimizer_arguments;
      ar & data.optimizer_map_file;
      ar & data.optimizer_lib_file;

      ar & data.simulator_logfile;
      ar & data.simulator_command;
      ar & data.simulator_arguments;
    }

    template< class Archive >
    inline
    void
    serialize( Archive &ar, message::question::InitSimulator &data, const unsigned int version ) {
      ar & data.simulator_logfile;
      ar & data.simulator_command;
      ar & data.simulator_arguments;
    }

    template< class Archive >
    inline
    void
    serialize( Archive &ar, message::question::InitCombined &data, const unsigned int version ) {
      ar & data.optimizer_logfile;
      ar & data.optimizer_library;
      ar & data.optimizer_arguments;
      ar & data.optimizer_map_file;
      ar & data.optimizer_lib_file;

      ar & data.simulator_logfile;
      ar & data.simulator_command;
      ar & data.simulator_arguments;
    }

    template< class Archive >
    inline
    void
    serialize( Archive &ar, message::question::GetPolicy &data, const unsigned int version ) {
      // Nothing.
    }

    template< class Archive >
    inline
    void
    serialize( Archive &ar, message::question::Simulate &data, const unsigned int version ) {
      ar & data.policy;
    }

    template< class Archive >
    inline
    void
    serialize( Archive &ar, message::question::Update &data, const unsigned int version ) {
      ar & data.policy;
      ar & data.reward;
    }

    template< class Archive >
    inline
    void
    serialize( Archive &ar, message::question::StepCombined &data, const unsigned int version ) {
      // Nothing.
    }

    template< class Archive >
    inline
    void
    serialize( Archive &ar, message::question::DumpOptimizer &data, const unsigned int version ) {
      ar & data.file;
      ar & data.mode;
    }

    template< class Archive >
    inline
    void
    serialize( Archive &ar, message::question::DumpPolicies &data, const unsigned int version ) {
      ar & data.file;
      ar & data.count;
      ar & data.display;
    }

    template< class Archive >
    inline
    void
    serialize( Archive &ar, message::question::Shutdown &data, const unsigned int version ) {
      // Nothing.
    }

    template< class Archive >
    inline
    void
```

```
154     serialize( Archive &ar, message::Question &question, const
    unsigned int version ) {
155       ar & question.data;
156     }
157
158   }
159 }
160
161
162 namespace boost {
163   namespace serialization {
164
165     template< class Archive >
166     inline
167     void
168     serialize( Archive &ar, message::answer::StartLogging &data, const
    unsigned int version ) {
169       // Nothing.
170     }
171
172     template< class Archive >
173     inline
174     void
175     serialize( Archive &ar, message::answer::InitOptimizer &data,
    const unsigned int version ) {
176       // Nothing.
177     }
178
179     template< class Archive >
180     inline
181     void
182     serialize( Archive &ar, message::answer::InitSimulator &data,
    const unsigned int version ) {
183       // Nothing.
184     }
185
186     template< class Archive >
187     inline
188     void
189     serialize( Archive &ar, message::answer::InitCombined &data, const
    unsigned int version ) {
190       // Nothing.
191     }
192
193     template< class Archive >
194     inline
195     void
196     serialize( Archive &ar, message::answer::GetPolicy &data, const
    unsigned int version ) {
197       ar & data.policy;
198     }
199
200     template< class Archive >
201     inline
202     void
203     serialize( Archive &ar, message::answer::Simulate &data, const
    unsigned int version ) {
204       ar & data.reward;
205     }
206
207     template< class Archive >
208     inline
209     void
210     serialize( Archive &ar, message::answer::Update &data, const
    unsigned int version ) {
211       // Nothing.
212     }
213
214     template< class Archive >
215     inline
216     void
217     serialize( Archive &ar, message::answer::StepCombined &data, const
    unsigned int version ) {
218       ar & data.got_policy;
219     }
220
221     template< class Archive >
222     inline
223     void
224     serialize( Archive &ar, message::answer::DumpOptimizer &data,
    const unsigned int version ) {
225       // Nothing.
226     }
227
228     template< class Archive >
229     inline
230     void
231     serialize( Archive &ar, message::answer::DumpPolicies &data, const
    unsigned int version ) {
232       // Nothing.
233     }
234
235     template< class Archive >
236     inline
237     void
238     serialize( Archive &ar, message::answer::Shutdown &data, const
    unsigned int version ) {
239       ar & data.done;
240     }
241
242     template< class Archive >
243     inline
244     void
245     serialize( Archive &ar, message::answer::Failure &data, const
    unsigned int version ) {
246       ar & data.what;
247     }
248
249     template< class Archive >
250     inline
251     void
252     serialize( Archive &ar, message::Answer &answer, const unsigned
    int version ) {
253       ar & answer.data;
254     }
255
256   }
257 }
```

Listing B.38: src/optimizer.hpp

```
#ifndef DICON_OPTIMIZER_HPP_
#define DICON_OPTIMIZER_HPP_

/*---- [Distribution] ------------------------------------------------
 * This file is part of the Disease Control System DiCon.
 *
 * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
 * Designed and developed with the guidance of Nedialko B. Dimitrov
 * and Lauren Ancel Meyers at the University of Texas at Austin.
 *
 * DiCon is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * DiCon is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
 *--------------------------------------------------------------------*/

/**
 * @file
 * @brief Optimizer class.
 */
#include "library.hpp"
#include "redirect.hpp"
#include "simulator.hpp"
#include <boost/bimap.hpp>


/**
 * @brief %Optimizer library calls.
 *
 * The optimizer namespace contains type definitions for all
 * %optimizer related calls. An %optimizer library to be used by the
 * system must implement the following functions and export them by
 * the following symbol names.
 *
 * - @c initialize(): see @link initialize_t() initialize_t@endlink.
 * - @c get_policy(): see @link get_policy_t() get_policy_t@endlink.
 * - @c update(): see @link update_t() update_t@endlink.
 * - @c policies(): see @link policies_t() policies_t@endlink.
 * - @c dump_state(): see @link dump_state_t() dump_state_t@endlink.
 * - @c get_error(): see @link get_error_t() get_error_t@endlink.
 *
 * Some of these functions must return data of arbitrary size (@c
 * get_policy(), @c policies()) where the caller cannot know in
 * advance how much data to expect. In these cases, the %optimizer
 * library must provide the required memory space and return a pointer
 * to that data, which is typically a 0-terminated list of nodes in
 * the policy tree.  The %optimizer wrapper is guaranteed not to
 * modify this memory, as indicated by the @c const specifier. The
 * same applies to the @c children() callback exported by the
 * %optimizer wrapper: it will return the children list through a
 * pointer to memory that is owned by the %optimizer wrapper; in turn,
 * the %optimizer library guarantees not to modify this memory.
 *
 * All functions except @c get_error() return an @c int return value.
 * This value is 0 if and only if the call was successful. Errors are
 * indicated by return values other than 0. In this case, the function
 * @c get_error() can be used to get a textual description of the
 * actual error.
 *
 * @see Optimizer class, Simulator class.
 */
namespace optimizer {

  /**
   * @brief Policy node.
   *
   * Policy node used by the %optimizer library. The %optimizer
   * library works only on @c int nodes which are translated by the
   * %optimizer wrapper to the actual values as returned by the
   * simulator's @c children() callback.  Lists of policy nodes (as
   * used by @c get_policy(), @c update(), @c policies()) are
   * 0-terminated; regular node values are positive integers within
   * the range of the @c int type (on the platform used to compile
   * both the system and the %optimizer library).
   */
  typedef int node_t;

  extern "C" {
    /**
     * @brief Children callback.
     *
     * Children callback exported to the %optimizer on initialization
     * (by the means of @c initialize()). The %optimizer library can
     * use this callback during any call to get the list of children
     * at any node in the policy tree as defined by the simulator
     * program associated with the corresponding Optimizer object
     * wrapping this %optimizer library.
     *
     * @param path 0-terminated path to node in policy tree (empty
     *   path for root node).
     * @param children 0-terminated list of child nodes returned by
     *   the %optimizer wrapper.
     */
    typedef int (*children_t)( const node_t *path, const node_t **
children );


    /**
     * @brief Initialize %optimizer.
     *
     * Initialize the %optimizer with the given arguments and children
     * callback, optionally restoring an %optimizer state previously
     * dumped with @c dump_state(). The %optimizer command-line
     * argument array @e argv contains as first element the name of
     * the %optimizer library itself, i.e., @e argc is always greater
     * than or equal to 1.
     *
     * This function is guaranteed to be called only once, and before
     * any of the other funtions is called.
     *
     * @param argc Number of elements in @e argv array.
     * @param argv %Optimizer command-line arguments.
     * @param children @c children() callback provided by %optimizer
```

```
121       *   wrapper.
122       * @param state_file  If not equal to @c NULL, restore %optimizer
123       *   state from this file.
124       *
125       * @returns 0 iff the call was successful.
126       */
127      typedef int (*initialize_t)( int argc, const char *const *argv,
128                                   children_t children, const char *
    state_file     );
129      /**
130       * @brief  Get next policy from %optimizer.
131       *
132       * Get the next policy from the %optimizer.  The %optimizer
133       * returns in @e policy the pointer to a 0-terminated policy node
134       * list indicating the next policy, or @c NULL when no policy is
135       * available at the moment. If the latter is indicated, a new
136       * policy must only become available after at least one call to @c
137       * update(), using one of the policies previously returned by @c
138       * get_policy(). Therefore, when no policies are pending and @c
139       * get_policy() returns no policy, the end of the optimization job
140       * has been reached.
141       *
142       * @param policy 0-terminated policy returned by the %optimizer
143       *   library, or @c NULL.
144       *
145       * @returns 0 iff the call was successful.
146       */
147      typedef int (*get_policy_t)( const node_t **policy
            );
148      /**
149       * @brief Update policy's reward value.
150       *
151       * Update the reward value of a policy previously returned by @c
152       * get_policy(). The reward value is always between 0 and 1.
153       *
154       * This function is guaranteed to be called only once for each
155       * policy returned by @c get_policy().
156       *
157       * @param policy 0-terminated policy.
158       * @param reward Policy's reward value.
159       *
160       * @returns 0 iff the call was successful.
161       */
162      typedef int (*update_t    )( const node_t *policy, double reward
            );
163      /**
164       * @brief Get list of best policies.
165       *
166       * Get the list of up to @e count best policies, based on the
167       * average reward value of each policy.  The %optimizer library is
168       * allowed to return less than the given number of policies (even
169       * none).  The number of policies actually available will be
170       * returned in @e count, and @e policies will contain a list of @e
171       * count consecutive 0-terminated policies, and @e rewards will
172       * contain a list of @e count consecutive average reward values
173       * for each corresponding policy.
174       *
175       * This function is guaranteed to be called only when no policies
176       * are pending, i.e., after calling @c update() once for each
177       * policy returned by @c get_policy().
178       *
179       * @param count  Maximum number of policies to return as well as
180       *   actual number of policies returned.
181       * @param policies  List of @e count consecutive 0-terminated
182       *   policies returned by the %optimizer library.
183       * @param rewards  List of @e count consecutive reward values
184       *   returned by the %optimizer library.
185       *
186       * @returns 0 iff the call was successful.
187       */
188      typedef int (*policies_t  )( unsigned *count,
189                                   const node_t **policies, const double
    **rewards );
190      /**
191       * @brief Dump internal %optimizer state.
192       *
193       * Dump the %optimizer's internal state, so that it can be
194       * completely restored by a later call to @c initialize(). There
195       * are no restrictions on the file format used.
196       *
197       * This function is guaranteed to be called only when no policies
198       * are pending, i.e., after calling @c update() once for each
199       * policy returned by @c get_policy().
200       *
201       * @param filename Name of dump file to create.
202       *
203       * @returns 0 iff the call was successful.
204       */
205      typedef int (*dump_state_t)( const char *filename
            );
206
207      /**
208       * @brief Get description of last error.
209       *
210       * Get a textual description of the last error.
211       *
212       * This function is guaranteed to be called only right after
213       * another function indicated an error by returning a value
214       * different from 0.
215       *
216       * @returns Textual description of the last error.
217       */
218      typedef const char * (*get_error_t)();
219    }
220
221  }
222
223
224  /// Errinfo storing slave error %message.
225  DICON_ERRINFO( slave_error, std::string );
226
227
228  /// %Optimizer related error.
229  struct OptimizerError : virtual Error
230  { virtual const char *what() const throw() { return "Optimizer_related
    _error."; } };
231
232  /// Policy node related error in %optimizer.
233  struct OptimizerNodeError : virtual OptimizerError
234  { virtual const char *what() const throw() { return "Policy_node_
    related_error_in_optimizer."; } };
235
```

```
236 /// Policy node not in %optimizer's translation table.
237 struct OptimizerUnknownNodeError : virtual OptimizerNodeError
238 { virtual const char *what() const throw() { return "Policy node not ↩
    in optimizer's translation table."; } };
239
240 /// Reached node limit in %optimizer's translation table.
241 struct OptimizerTooManyNodesError : virtual OptimizerNodeError
242 { virtual const char *what() const throw() { return "Reached node ↩
    limit in optimizer's translation table."; } };
243
244
245 /// Calling %optimizer function failed.
246 struct OptimizerSlaveError : virtual OptimizerError
247 { virtual const char *what() const throw() { return "Calling optimizer ↩
    function failed."; } };
248
249 /// Calling %optimizer's initialize() failed.
250 struct OptimizerInitializeError : virtual OptimizerSlaveError
251 { virtual const char *what() const throw() { return "Calling optimizer ↩
    's initialize() failed."; } };
252
253 /// Calling %optimizer's get_policy() failed.
254 struct OptimizerGetPolicyError : virtual OptimizerSlaveError
255 { virtual const char *what() const throw() { return "Calling optimizer ↩
    's get_policy() failed."; } };
256
257 /// Calling %optimizer's update() failed.
258 struct OptimizerUpdateError : virtual OptimizerSlaveError
259 { virtual const char *what() const throw() { return "Calling optimizer ↩
    's update() failed."; } };
260
261 /// Calling %optimizer's policies() failed.
262 struct OptimizerPoliciesError : virtual OptimizerSlaveError
263 { virtual const char *what() const throw() { return "Calling optimizer ↩
    's policies() failed."; } };
264
265 /// Calling %optimizer's dump_state() failed.
266 struct OptimizerDumpStateError : virtual OptimizerSlaveError
267 { virtual const char *what() const throw() { return "Calling optimizer ↩
    's dump_state() failed."; } };
268
269
270 class Combined;
271
272 namespace detail {
273   extern "C" int optimizer_children( const optimizer::node_t *, const ↩
    optimizer::node_t ** );
274 }
275
276 /**
277  * @brief %Optimizer wrapper.
278  *
279  * The Optimizer class wraps an %optimizer library. It provides
280  * translation between the policies returned by a simulator's @c
281  * children() callback to @c int values used by the optimization
282  * algorithm defined in the wrapped %optimizer library. It also
283  * handles wrapping and unwrapping of C++ lists to C arrays and
284  * provides the necessary memory management.
285  *
286  * The %optimizer library must export symbols as shown in the
287  * description of the optimizer namespace.
288  *
289  * @note Only a single Optimizer instance is allowed to exist at any
290  *   time. If another instance is created before the previous
291  *   instance is destroyed, the constructor of the second instance
292  *   will fail.
293  */
294 class Optimizer
295   : boost::noncopyable
296 {
297   friend class Combined;
298   friend int detail::optimizer_children( const optimizer::node_t *, ↩
    const optimizer::node_t ** );
299
300 public:
301   /**
302    * @brief Load %optimizer library and simulator.
303    *
304    * Constructor that initializes the given %optimizer library and
305    * simulator. The simulator is used exclusively for querying the
306    * policy space through the simulator's @c children() callback. Both
307    * %optimizer and simulator may have logfiles associated that are
308    * used to record messages written to standard error output (in case
309    * of the %optimizer, also standard output) by both %optimizer
310    * library and simulator program.
311    *
312    * If either logfile given by @e optimizer_logfile or @e
313    * simulator_logfile exits, it will not be overwritten. Instead, a
314    * new name that does not exist will be chosen according to
315    * File::unique().
316    *
317    * Both %optimizer library arguments and simulator command arguments
318    * should contain only the actual arguments, without the name of the
319    * library or program as first argument.
320    *
321    * @param optimizer_logfile Logfile to record %optimizer library's
322    *   output to standard output/error (or empty path for no logging).
323    * @param simulator_logfile Logfile to record simulator program's
324    *   output to standard error (or empty path for no logging).
325    * @param optimizer_library Path to %optimizer library.
326    * @param optimizer_arguments %Optimizer library arguments.
327    * @param simulator_command Path to simulator command.
328    * @param simulator_arguments %Simulator command arguments.
329    * @param optimizer_map_file Path to %optimizer map dump file (empty
330    *   if %optimizer map state should not be restored).
331    * @param optimizer_lib_file Path to %optimizer lib dump file (empty
332    *   if %optimizer lib state should not be restored).
333    *
334    * @throws LibraryOpenError when the %optimizer library cannot be
335    *   opened.
336    * @throws LibrarySymbolError when not all required symbols are
337    *   exported by the %optimizer library.
338    * @throws AssertionError when another Optimizer instance still
339    *   exists.
340    * @throws RedirectorError when the %optimizer library logfile
341    *   cannot be setup.
342    * @throws FileReadError when the %optimizer map dump file cannot be
343    *   read.
344    * @throws OptimizerInitializeError when initializing the %optimizer
345    *   library (@c initialize()) failed.
346    */
347   Optimizer( const boost::filesystem::path &optimizer_logfile
```

```
348             , const boost::filesystem::path &simulator_logfile
349             , const std::string &optimizer_library, const arguments_t &optimizer_arguments
350             , const std::string &simulator_command, const arguments_t &simulator_arguments
351             , const boost::optional<boost::filesystem::path> &optimizer_map_file
352             , const boost::optional<boost::filesystem::path> &optimizer_lib_file
353             );
354   /**
355    * @brief Free %optimizer library and simulator.
356    *
357    * Destructor that frees the %optimizer library and simulator
358    * program. This releases all resources and closes the logfiles used
359    * for %optimzier and simulator output
360    */
361   ~Optimizer();
362
363 public:
364   /**
365    * @brief Get next policy.
366    *
367    * Get the next available policy from the %optimizer. If no policy
368    * is available right now, this is indicated by returning an empty
369    * object.
370    *
371    * @return Next available policy, or an empty object when no policy
372    *   is available (until update() is called at least once).
373    *
374    * @throws OptimizerGetPolicyError when getting a policy from the
375    *   %optimizer library (<code>%get_policy()</code>) failed.
376    */
377   boost::optional<policy_t> get_policy();
378   /**
379    * @brief Update policy's reward.
380    *
381    * Update the reward value of the given policy. The policy must have
382    * been returned by a previous call to get_policy(). For each
383    * policy, update() must only be called once.
384    *
385    * @param policy Policy.
386    * @param reward Reward.
387    *
388    * @throws AssertionError when no policy is pending or reward value
389    *   is less than 0 or greater than 1.
390    * @throws OptimizerUpdateError when updating policy's reward
391    *   (<code>%update()</code>) in the %optimizer library failed.
392    */
393   void update( const policy_t &policy, double reward );
394
395   /**
396    * @brief Dump %optimizer map state.
397    *
398    * Dump the %optimizer's one-to-one correspondence map between
399    * policies returned by the simulator's @c children() callback and
400    * the @c int values handed over to the %optimizer library. If a
401    * file with the given name already exists, it will be overwritten.
402    * This method must only be called when no policies are pending.
403    *
404    * @param file Filename of the map dump file.
405    *
406    * @throws AssertionError when at least one policy is pending.
407    * @throws FileWriteError when writing to the %optimizer map dump
408    *   file failed.
409    */
410   void dump_optimizer_map( const boost::filesystem::path &file );
411   /**
412    * @brief Dump %optimizer lib state.
413    *
414    * Dump the %optimizer library's internal state. If a file with the
415    * given name already exists, it will be overwritten. This must only
416    * be called when no policies are pending.
417    *
418    * @param file Filename of the lib dump file.
419    *
420    * @throws AssertionError when at least one policy is pending.
421    * @throws OptimizerDumpStateError when dumping the %optimizer
422    *   library's internal state (<code>%dump_state()</code>) failed.
423    *
424    */
425   void dump_optimizer_lib( const boost::filesystem::path &file );
426   /**
427    * @brief Dump list of best policies.
428    *
429    * Dump the list of up to @e count best policies, as given by each
430    * policy's average reward value. The %optimizer library might
431    * decide to dump less than @e count policies (even none). The dump
432    * can either by textual or binary, where binary output means that
433    * policies are dumped as they are returned by the simulator's @c
434    * children() callback, as a list of nodes describing the actual
435    * policy path, whereas textual output uses the simulator's @c
436    * display() callback to create a human-readable representation of
437    * each policy.
438    *
439    * In both cases the dump file will contain two lines per policy,
440    * with the average reward followed by a description of the
441    * policy. This description is either an escaped version of the text
442    * returned by the simulator's @c display() callback, or a
443    * comma-separated list of the raw policy nodes as returned by the
444    * simulator's @c children() callback, also escaped.
445    *
446    * The escape method used is similar to the quoted-printable
447    * encoding; see the description of the qp_encode() function for
448    * details.
449    *
450    * An example (binary) policy dump might look like the following.
451    *
452    * @verbatim
453 0.960000000000
454 =80=02K=04.,=80=02K=04.
455 @endverbatim
456    *
457    * Here, a single policy was dumped, with an average reward value of
458    * 0.96. The policy is described by a path of length 2 in the
459    * policy tree, where each path element is equivalent to the
460    * 5-character string given as hexadecimal <code>0x80 0x02 0x4b 0x04
461    * 0x2e</code>. It is up to the simulator what this means. A textual
462    * dump of the same policy (taken at the same %optimizer state)
463    * looked like the following.
464    *
465    * @verbatim
```

```
0.960000000000
[4, 4]
@endverbatim
   *
   * Here, the entire text <code>[4, 4]</code> was returned by the
   * simulator's @c display() callback. As all the characters in this
   * string are printable, escaping was not necessary.
   *
   * This method must only be called when no policies are pending.
   *
   * @param file Name of dump file to create.
   * @param count Maximum number of policies to dump.
   * @param display @c true iff textual dump output is to be used.
   *
   * @throws AssertionError when at least one policy is pending.
   * @throws FileWriteError when writing to the policies dump file
   *   failed.
   * @throws OptimizerPoliciesError when getting the list of policies
   *   from the %optimizer library (<code>%policies()</code>) failed.
   */
  void dump_policies( const boost::filesystem::path &file, unsigned count, bool display );

private:
  void initialize( const std::string &name, const arguments_t &arguments,
                   optimizer::children_t children, const boost::optional<boost::filesystem::path> &state_file );

private:
  void children_c( const optimizer::node_t *path, const optimizer::node_t **children );

  typedef std::vector<std::pair<policy_t, double> > policies_result_type;

  policies_result_type policies( unsigned count );

  template< typename T >
  void load_symbol( T &variable, const std::string &symbol );

  policy_t id_to_name( const optimizer::node_t *list );
  const optimizer::node_t *name_to_id( const policy_t &policy );

private:
  Simulator &simulator();

private:
  DynamicLibrary library_;
  Simulator simulator_;

  typedef unsigned policy_count_t;
  policy_count_t policy_count_;

  boost::scoped_ptr<Redirector> stdout_;
  boost::scoped_ptr<Redirector> stderr_;

  optimizer::initialize_t initialize_;
  optimizer::get_policy_t get_policy_;
  optimizer::update_t     update_;
  optimizer::policies_t   policies_;
  optimizer::dump_state_t dump_state_;
  optimizer::get_error_t  get_error_;

  std::vector<optimizer::node_t> temporary_;

private:
  void load_state( std::istream &in );
  void save_state( std::ostream &out );

  void load_state( const std::string &file );
  void save_state( const std::string &file );

  struct id {};
  struct name {};

  typedef boost::bimap< boost::bimaps::tagged<std::string       , name>
                      , boost::bimaps::tagged<optimizer::node_t, id  >
                      > nodes_t;
  nodes_t nodes_;
};


/**
 * @brief %Combined %optimizer/simulator wrapper.
 *
 * The Combined class wraps both an %optimizer and a simulator.
 * Similar to the Optimizer class it provides translation between the
 * policies returned by a simulator's @c children() callback to @c int
 * values used by the optimization algorithm defined in the wrapped
 * %optimizer library.
 *
 * In contrast to the regular %optimizer and simulator interfaces, it
 * provides only a single method to take a %step, i.e., get a policy
 * from the embedded %optimizer, simulate it in the embedded
 * simulator, and update the reward value accordingly. In addition,
 * only the regular @c dump_ methods are supported.
 */
class Combined
  : boost::noncopyable
{
public:
  /**
   * @brief Load optimizer library and simulator.
   *
   * Constructor that initializes the given %optimizer library and
   * simulator. The simulator is used both for querying the policy
   * space through the simulator's @c children() callback, and for
   * simulating policies through the simulator's @c simulate()
   * callback. Both %optimizer and simulator may have logfiles
   * associated that are used to record messages written to standard
   * error output (in case of the %optimizer, also standard output) by
   * both %optimizer library and simulator program.
   *
   * If either logfile given by @e optimizer_logfile or @e
   * simulator_logfile exits, it will not be overwritten. Instead, a
   * new name that does not exist will be chosen according to
   * File::unique().
   *
   * Both %optimizer library arguments and simulator command arguments
   * should contain only the actual arguments, without the name of the
   * library or program as first argument.
```

```
583     *
584     * @param optimizer_logfile  Logfile to record %optimizer library's
585     *   output to standard output/error (or empty path for no logging).
586     * @param simulator_logfile Logfile to record simulator program's
587     *   output to standard error (or empty path for no logging).
588     * @param optimizer_library Path to %optimizer library.
589     * @param optimizer_arguments %Optimizer library arguments.
590     * @param simulator_command Path to simulator command.
591     * @param simulator_arguments %Simulator command arguments.
592     * @param optimizer_map_file Path to %optimizer map dump file (empty
593     *   if %optimizer map state should not be restored).
594     * @param optimizer_lib_file Path to %optimizer lib dump file (empty
595     *   if %optimizer lib state should not be restored).
596     *
597     * @throws LibraryOpenError when the %optimizer library cannot be
598     *   opened.
599     * @throws LibrarySymbolError when not all required symbols are
600     *   exported by the %optimizer library.
601     * @throws AssertionError when another Optimizer instance still
602     *   exists.
603     * @throws RedirectorError when the %optimizer library logfile
604     *   cannot be setup.
605     * @throws FileReadError when the %optimizer map dump file cannot be
606     *   read.
607     * @throws OptimizerInitializeError when initializing the %optimizer
608     *   library (@c initialize()) failed.
609     */
610   Combined( const boost::filesystem::path &optimizer_logfile
611           , const boost::filesystem::path &simulator_logfile
612           , const std::string &optimizer_library, const arguments_t &
    optimizer_arguments
613           , const std::string &simulator_command, const arguments_t &
    simulator_arguments
614           , const boost::optional<boost::filesystem::path> &
    optimizer_map_file
615           , const boost::optional<boost::filesystem::path> &
    optimizer_lib_file
616           );
617
618 public:
619   /**
620     * @brief Dump %optimizer map state.
621     *
622     * @see Optimizer::dump_optimizer_map().
623     */
624   void dump_optimizer_map( const boost::filesystem::path &file );
625   /**
626     * @brief Dump %optimizer lib state.
627     *
628     * @see Optimizer::dump_optimizer_lib().
629     */
630   void dump_optimizer_lib( const boost::filesystem::path &file );
631   /**
632     * @brief Dump list of best policies.
633     *
634     * @see Optimizer::dump_policies().
635     */
636   void dump_policies( const boost::filesystem::path &file, unsigned
    count, bool display );
637
638 public:
639   /**
640     * @brief Take step in node.
641     *
642     * Take another step in the combined %optimizer/simulator node. This
643     * will try to get a new policy from the embedded %optimizer,
644     * simulate it using the embedded simulator, and update the return
645     * value at the %optimizer accordingly. This method returns @c
646     * false if and only if no policy is available from the %optimizer.
647     *
648     * @returns @c true iff a next method was available.
649     *
650     * @throws OptimizerGetPolicyError when getting a policy from the
651     *   %optimizer library (<code>%get_policy()</code>) failed.
652     * @throws SimulatorSimulateError when simulating the policy in
653     *   simulator (<code>%simulate()</code>) failed.
654     * @throws SimulatorInvalidRewardError when the reward value was
655     *   less than 0 or greater than 1.
656     * @throws OptimizerUpdateError when updating policy's reward
657     *   (<code>%update()</code>) in the %optimizer library failed.
658     */
659   bool step();
660
661 private:
662   Optimizer optimizer_;
663 };
664
665 #endif //DICON_OPTIMIZER_HPP_
```

Listing B.39: src/optimizer.cpp

```
1  /*---- [Distribution] ------------------------------------------------
2   * This file is part of the Disease Control System DiCon.
3   *
4   * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
5   * Designed and developed with the guidance of Nedialko B. Dimitrov
6   * and Lauren Ancel Meyers at the University of Texas at Austin.
7   *
8   * DiCon is free software: you can redistribute it and/or modify it
9   * under the terms of the GNU General Public License as published by
10  * the Free Software Foundation, either version 3 of the License, or
11  * (at your option) any later version.
12  *
13  * DiCon is distributed in the hope that it will be useful, but
14  * WITHOUT ANY WARRANTY; without even the implied warranty of
15  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
16  * General Public License for more details.
17  *
18  * You should have received a copy of the GNU General Public License
19  * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
20  *--------------------------------------------------------------------*/
21
22 #include "optimizer.hpp"
23 #include "log.hpp"
24 #include "util.hpp"
25 #include <boost/archive/binary_iarchive.hpp>
26 #include <boost/archive/binary_oarchive.hpp>
27 #include <boost/filesystem/operations.hpp>
28 #include <boost/foreach.hpp>
29 #include <boost/format.hpp>
```

```
30 #include <boost/scoped_ptr.hpp>
31 #include <fstream>
32
33
34 namespace detail {
35
36   static Optimizer *optimizer_instance = NULL;
37
38 }
39
40
41
42 namespace detail {
43
44   extern "C"
45   int
46   optimizer_children( const optimizer::node_t *path, const optimizer::node_t **children ) {
47     try {
48       try {
49         if( detail::optimizer_instance == NULL )
50           BOOST_THROW_EXCEPTION( AssertionError() );
51
52         detail::optimizer_instance->children_c( path, children );
53         return 0;
54       }
55       catch( boost::exception &e ) {
56         log_error( (boost::format("Optimizer_children_callback_failed.\n%s")
57                     % boost::diagnostic_information(e)).str() );
58         return -1;
59       }
60     }
61     catch( ... ) {
62       return -99;
63     }
64   }
65
66 }
67
68
69
70 template< typename T >
71 void
72 Optimizer::load_symbol( T &variable, const std::string &symbol ) {
73   *reinterpret_cast<void **>(&variable) = library_[symbol];
74 }
75
76
77 Optimizer::Optimizer( const boost::filesystem::path &optimizer_logfile
78                     , const boost::filesystem::path &simulator_logfile
79                     , const std::string &optimizer_library, const arguments_t &optimizer_arguments
80                     , const std::string &simulator_command, const arguments_t &simulator_arguments
81                     , const boost::optional<boost::filesystem::path> &optimizer_map_file
82                     , const boost::optional<boost::filesystem::path> &optimizer_lib_file
83                     )
84   : library_(optimizer_library), simulator_(simulator_logfile, simulator_command, simulator_arguments),
85     policy_count_(0)
86 {
87   if( detail::optimizer_instance != NULL )
88     BOOST_THROW_EXCEPTION( AssertionError() );
89
90   if( !optimizer_logfile.empty() ) {
91     File file = File::unique( optimizer_logfile );
92     stdout_.reset( new Redirector(file.file(), STDOUT_FILENO) );
93     stderr_.reset( new Redirector(file.file(), STDERR_FILENO) );
94   }
95
96   if( optimizer_map_file )
97     load_state( optimizer_map_file->file_string() );
98
99   try {
100    detail::optimizer_instance = this;
101
102    load_symbol( initialize_, "initialize" );
103    load_symbol( get_policy_, "get_policy" );
104    load_symbol( update_    , "update"     );
105    load_symbol( policies_  , "policies"   );
106    load_symbol( dump_state_, "dump_state" );
107    load_symbol( get_error_ , "get_error"  );
108
109    initialize( optimizer_library, optimizer_arguments,
110                detail::optimizer_children, optimizer_lib_file );
111  }
112  catch( ... ) {
113    // Make sure to unregister instance.
114    detail::optimizer_instance = NULL;
115    throw;
116  }
117 }
118
119
120 Optimizer::~Optimizer() {
121   assert( detail::optimizer_instance == this );
122   detail::optimizer_instance = NULL;
123 }
124
125
126 void
127 Optimizer::load_state( std::istream &in ) {
128   boost::archive::binary_iarchive( in )
129     >> boost::serialization::make_nvp( "nodes", nodes_ );
130 }
131
132
133 void
134 Optimizer::save_state( std::ostream &out ) {
135   boost::archive::binary_oarchive( out )
136     << boost::serialization::make_nvp( "nodes", nodes_ );
137 }
138
139
140 void
141 Optimizer::load_state( const std::string &file ) {
142   std::ifstream in( file.c_str() );
143   if( in.fail() )
```

```
144     BOOST_THROW_EXCEPTION( FileReadError() << errinfo_file_name(file) );
145   load_state( in );
146 }
147
148
149 void
150 Optimizer::save_state( const std::string &file ) {
151   std::ofstream out( file.c_str() );
152   save_state( out );
153   if( out.fail() )
154     BOOST_THROW_EXCEPTION( FileWriteError() << errinfo_file_name(file) );
155 }
156
157
158 policy_t
159 Optimizer::id_to_name( const optimizer::node_t *const list ) {
160   std::vector<std::string> policy;
161
162   for( size_t i = 0; list[i]; ++i ) {
163     nodes_t::map_by<id>::const_iterator it;
164
165     if( (it = nodes_.by<id>().find(list[i])) == nodes_.by<id>().end() )
166       BOOST_THROW_EXCEPTION( OptimizerUnknownNodeError() << errinfo_value<optimizer::node_t>(list[i]) );
167
168     policy.push_back( it->second );
169   }
170
171   return policy;
172 }
173
174
175 const optimizer::node_t *
176 Optimizer::name_to_id( const policy_t &policy ) {
177   temporary_.clear();
178
179   BOOST_FOREACH( const std::string &node, policy ) {
180     nodes_t::map_by<name>::const_iterator it;
181
182     if( (it = nodes_.by<name>().find(node)) == nodes_.by<name>().end() ) {
183       size_t new_id = nodes_.size() + 1;
184
185       if( new_id >= size_t(std::numeric_limits<optimizer::node_t>::max()) ) {
186         BOOST_THROW_EXCEPTION( OptimizerTooManyNodesError() << errinfo_value<size_t>(nodes_.size())
187                                << errinfo_max_value<size_t>(size_t(std::numeric_limits<optimizer::node_t>::max())-1) );
188       }
189
190       it = nodes_.by<name>().insert( std::make_pair(node, new_id) ).first;
191     }
192
193     assert( it->second != 0 );
194     temporary_.push_back( it->second );
195   }
196
197   temporary_.push_back( 0 );
198   return temporary_.data();
199 }
200
201
202 void
203 Optimizer::children_c( const optimizer::node_t *const path, const optimizer::node_t **const children ) {
204   *children = name_to_id(simulator_.children( id_to_name(path) ));
205 }
206
207
208 boost::optional<policy_t>
209 Optimizer::get_policy() {
210   assert( policy_count_ <= std::numeric_limits<policy_count_t>::max() );
211   if( policy_count_ == std::numeric_limits<policy_count_t>::max() )
212     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<policy_count_t>(policy_count_) );
213
214   const optimizer::node_t *policy;
215
216   // This call to get_policy_() might in turn (and probably will in most
217   // cases) result in the library indirectly calling children_c() above.
218
219   if( get_policy_(&policy) != 0 )
220     BOOST_THROW_EXCEPTION( OptimizerGetPolicyError() << errinfo_slave_error(get_error_()) );
221
222   assert( policy_count_ < std::numeric_limits<policy_count_t>::max() );
223
224   if( policy != NULL ) {
225     ++policy_count_;  // count
226     return id_to_name( policy );
227   }
228   else
229     return boost::none;
230 }
231
232
233 void
234 Optimizer::update( const policy_t &policy, double reward ) {
235   if( policy_count_ == 0 )
236     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<policy_count_t>(policy_count_) );
237
238   DICON_ASSERT_RANGE<double>( reward, 0, 1 );
239
240   if( update_(name_to_id(policy), reward) != 0 )
241     BOOST_THROW_EXCEPTION( OptimizerUpdateError() << errinfo_slave_error(get_error_()) );
242
243   assert( policy_count_ > 0 );
244   --policy_count_;
245 }
246
247
```

```
248 Optimizer::policies_result_type
249 Optimizer::policies( unsigned count ) {
250   if( policy_count_ != 0 )
251     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<policy_count_t>(policy_count_) );
252
253   const optimizer::node_t *policies;
254   const double *rewards;
255
256   if( policies_(&count, &policies, &rewards) != 0 )
257     BOOST_THROW_EXCEPTION( OptimizerPoliciesError() << errinfo_slave_error(get_error_()) );
258
259   policies_result_type list;
260
261   size_t offset = 0;
262   for( unsigned i = 0; i < count; ++i ) {
263     const policy_t &policy
264       = id_to_name( policies+offset );
265
266     offset += (policy.size() + 1);
267     list.push_back( policies_result_type::value_type(policy, rewards[i]) );
268   }
269
270   return list;
271 }
272
273
274 void
275 Optimizer::dump_optimizer_map( const boost::filesystem::path &file ) {
276   if( policy_count_ != 0 )
277     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<policy_count_t>(policy_count_) );
278
279   boost::filesystem::path tmp_file = File::unique_temporary( file );
280
281   save_state( tmp_file.file_string() );
282
283   File::rename( tmp_file, file );
284 }
285
286
287 void
288 Optimizer::dump_optimizer_lib( const boost::filesystem::path &file ) {
289   if( policy_count_ != 0 )
290     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<policy_count_t>(policy_count_) );
291
292   boost::filesystem::path tmp_file = File::unique_temporary( file );
293
294   if( dump_state_(tmp_file.file_string().c_str()) != 0 )
295     BOOST_THROW_EXCEPTION( OptimizerDumpStateError() << errinfo_slave_error(get_error_()) );
296
297   File::rename( tmp_file, file );
298 }
299
300
301
302 namespace detail {
303
304   static
305   std::string
306   policy_to_str( const policy_t &policy ) {
307     std::string result;
308
309     bool first = true;
310     BOOST_FOREACH( const std::string &node, policy ) {
311       if( !first )
312         result += ',';
313       else
314         first = false;
315
316       result += qp_encode( node, "," );
317     }
318
319     return result;
320   }
321
322 }
323
324
325
326 void
327 Optimizer::dump_policies( const boost::filesystem::path &file, unsigned count, bool display ) {
328   if( policy_count_ != 0 )
329     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<policy_count_t>(policy_count_) );
330
331   boost::filesystem::path tmp_file = File::unique_temporary( file );
332
333   {
334     std::ofstream out( tmp_file.file_string().c_str() );
335
336     BOOST_FOREACH( const policies_result_type::value_type &entry, policies(count) ) {
337       const policy_t &policy = entry.first;
338       const double &reward = entry.second;
339
340       std::string str;
341
342       if( display )
343         str = qp_encode( simulator_.display(policy) );
344       else
345         str = detail::policy_to_str( policy );
346
347       out << (boost::format("%.12f") % reward) << std::endl
348           << str << std::endl;
349     }
350
351     if( out.fail() )
352       BOOST_THROW_EXCEPTION( FileWriteError() << errinfo_file_name(tmp_file.file_string()) );
353
354     out.close();
355   }
356
357   File::rename( tmp_file, file );
358 }
359
```

```
360
361 void
362 Optimizer::initialize( const std::string &name, const arguments_t &
    arguments,
363                        optimizer::children_t children, const boost::
    optional<boost::filesystem::path> &state_file )
364 {
365   std::vector<const char *> argv;
366   argv.push_back( name.c_str() );
367
368   BOOST_FOREACH( const std::string &arg, arguments ) {
369     argv.push_back( arg.c_str() );
370   }
371
372   if( initialize_(argv.size(), argv.data(), children,
373                   state_file ? state_file->file_string().c_str() :
    NULL) != 0 )
374   {
375     BOOST_THROW_EXCEPTION( OptimizerInitializeError() <<
    errinfo_slave_error(get_error_()) );
376   }
377 }
378
379
380 Simulator &
381 Optimizer::simulator() {
382   return simulator_;
383 }
384
385
386 Combined::Combined( const boost::filesystem::path &optimizer_logfile
387                   , const boost::filesystem::path &simulator_logfile
388                   , const std::string &optimizer_library, const
    arguments_t &optimizer_arguments
389                   , const std::string &simulator_command, const
    arguments_t &simulator_arguments
390                   , const boost::optional<boost::filesystem::path> &
    optimizer_map_file
391                   , const boost::optional<boost::filesystem::path> &
    optimizer_lib_file
392                   )
393   : optimizer_(optimizer_logfile, simulator_logfile,
394                optimizer_library, optimizer_arguments,
395                simulator_command, simulator_arguments,
396                optimizer_map_file, optimizer_lib_file)
397 {
398 }
399
400
401 void
402 Combined::dump_optimizer_map( const boost::filesystem::path &file ) {
403   optimizer_.dump_optimizer_map( file );
404 }
405
406
407 void
408 Combined::dump_optimizer_lib( const boost::filesystem::path &file ) {
409   optimizer_.dump_optimizer_lib( file );
410 }
411
412
```

```
413 void
414 Combined::dump_policies( const boost::filesystem::path &file, unsigned
    count, bool display ) {
415   optimizer_.dump_policies( file, count, display );
416 }
417
418
419 bool
420 Combined::step() {
421   const boost::optional<policy_t> &policy = optimizer_.get_policy();
422
423   if( !policy )
424     return false;
425
426   double reward = optimizer_.simulator().simulate( *policy );
427
428   optimizer_.update( *policy, reward );
429
430   return true;
431 }
```

Listing B.40: src/pipe.hpp

```
1  #ifndef DICON_PIPE_HPP_
2  #define DICON_PIPE_HPP_
3
4  /*---- [Distribution] ------------------------------------------------
5   * This file is part of the Disease Control System DiCon.
6   *
7   * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8   * Designed and developed with the guidance of Nedialko B. Dimitrov
9   * and Lauren Ancel Meyers at the University of Texas at Austin.
10  *
11  * DiCon is free software: you can redistribute it and/or modify it
12  * under the terms of the GNU General Public License as published by
13  * the Free Software Foundation, either version 3 of the License, or
14  * (at your option) any later version.
15  *
16  * DiCon is distributed in the hope that it will be useful, but
17  * WITHOUT ANY WARRANTY; without even the implied warranty of
18  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19  * General Public License for more details.
20  *
21  * You should have received a copy of the GNU General Public License
22  * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23  *--------------------------------------------------------------------*/
24
25 /**
26  * @file
27  * @brief Pipe class.
28  */
29 #include <boost/noncopyable.hpp>
30
31
32 /**
33  * @brief Unix pipeline.
34  *
35  * The Pipe class implements a Unix pipeline. This pipeline is created
36  * on construction of a Pipe object and can be accessed through Unix
```

```
 * file descriptors. Unless explicitly released, the object owns each
 * end of the pipeline and closes the corresponding file descriptors
 * when destroyed.
 */
class Pipe
  : boost::noncopyable
{
public:
  /**
   * @brief Create Unix pipeline.
   *
   * Constructor that creates a new Unix pipeline.
   *
   * @throws PipeError when the pipeline could not be created.
   */
  Pipe();
  /**
   * @brief Free Unix pipeline.
   *
   * Destructor that releases the Unix pipeline. Either endpoint of
   * the pipeline that has not been released through read_end(bool) or
   * write_end(bool) is closed.
   */
  ~Pipe();

public:
  /**
   * @brief Get read endpoint.
   *
   * Get the file descriptor for the read endpoint of the
   * pipeline. This does not release ownership of this endpoint.
   *
   * @returns %File descriptor for read endpoint of pipeline.
   * @throws FileNotOwnedError when object does not own read endpoint
   *   anymore.
   */
  int read_end() const;
  /**
   * @brief Get write endpoint.
   *
   * Get the file descriptor for the write endpoint of the
   * pipeline. This does not release ownership of this endpoint.
   *
   * @returns %File descriptor for write endpoint of pipeline.
   * @throws FileNotOwnedError when object does not own write endpoint
   *   anymore.
   */
  int write_end() const;

  /**
   * @brief Get read endpoint, possibly taking ownership.
   *
   * Get the file descriptor for the read endpoint of the pipeline.
   * When @e take_ownership is @c true, the object loses ownership of
   * the read endpoint. It is up to the caller to close the file
   * descriptor when it is not needed anymore.
   *
   * @return %File descriptor for read endpoint of pipeline.
   * @throws FileNotOwnedError when object does not own read endpoint
   *   anymore.
   */
  int read_end( bool take_ownership = false );
  /**
   * @brief Get write endpoint, possibly taking ownership.
   *
   * Get the file descriptor for the write endpoint of the pipeline.
   * When @e take_ownership is @c true, the object loses ownership of
   * the write endpoint. It is up to the caller to close the file
   * descriptor when it is not needed anymore.
   *
   * @return %File descriptor for write endpoint of pipeline.
   * @throws FileNotOwnedError when object does not own write endpoint
   *   anymore.
   */
  int write_end( bool take_ownership = false );

private:
  int pipefd_[2];
  bool owned_[2];
};

#endif //DICON_PIPE_HPP_
```

Listing B.41: src/pipe.cpp

```
/*--- [Distribution] ---------------------------------------------------
 * This file is part of the Disease Control System DiCon.
 *
 * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
 * Designed and developed with the guidance of Nedialko B. Dimitrov
 * and Lauren Ancel Meyers at the University of Texas at Austin.
 *
 * DiCon is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * DiCon is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
 *----------------------------------------------------------------------*/

#include "pipe.hpp"
#include "file.hpp"
#include <cerrno>


Pipe::Pipe() {
  if( pipe(pipefd_) != 0 )
    BOOST_THROW_EXCEPTION( PipeError() << errinfo_api_function("pipe")
 << errinfo_errno(errno) );

  owned_[0] = true;
  owned_[1] = true;
}

```

```
35
36 Pipe::~Pipe() {
37   if( owned_[0] ) close(pipefd_[0]);
38   if( owned_[1] ) close(pipefd_[1]);
39 }
40
41
42 int
43 Pipe::read_end() const {
44   if( !owned_[0] ) {
45     BOOST_THROW_EXCEPTION( FileNotOwnedError()
46                            << errinfo_file_descriptor(pipefd_[0]) );
47   }
48
49   return pipefd_[0];
50 }
51
52
53 int
54 Pipe::write_end() const {
55   if( !owned_[1] ) {
56     BOOST_THROW_EXCEPTION( FileNotOwnedError()
57                            << errinfo_file_descriptor(pipefd_[1]) );
58   }
59
60   return pipefd_[1];
61 }
62
63
64 int
65 Pipe::read_end( bool take_ownership ) {
66   if( !owned_[0] ) {
67     BOOST_THROW_EXCEPTION( FileNotOwnedError()
68                            << errinfo_file_descriptor(pipefd_[0]) );
69   }
70
71   if( take_ownership )
72     owned_[0] = false;
73
74   return pipefd_[0];
75 }
76
77
78 int
79 Pipe::write_end( bool take_ownership ) {
80   if( !owned_[1] ) {
81     BOOST_THROW_EXCEPTION( FileNotOwnedError()
82                            << errinfo_file_descriptor(pipefd_[1]) );
83   }
84
85   if( take_ownership )
86     owned_[1] = false;
87
88   return pipefd_[1];
89 }
```

Listing B.42: src/redirect.hpp

```
1 #ifndef DICON_REDIRECT_HPP_
2 #define DICON_REDIRECT_HPP_
3
4 /*--- [Distribution] ------------------------------------------------
5  * This file is part of the Disease Control System DiCon.
6  *
7  * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8  * Designed and developed with the guidance of Nedialko B. Dimitrov
9  * and Lauren Ancel Meyers at the University of Texas at Austin.
10 *
11 * DiCon is free software: you can redistribute it and/or modify it
12 * under the terms of the GNU General Public License as published by
13 * the Free Software Foundation, either version 3 of the License, or
14 * (at your option) any later version.
15 *
16 * DiCon is distributed in the hope that it will be useful, but
17 * WITHOUT ANY WARRANTY; without even the implied warranty of
18 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19 * General Public License for more details.
20 *
21 * You should have received a copy of the GNU General Public License
22 * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23 *-----------------------------------------------------------------*/
24
25 /**
26  * @file
27  * @brief Redirector class.
28  */
29 #include "file.hpp"
30
31
32 /**
33  * @brief Redirect files.
34  *
35  * The Redirector class implements reversible redirection of file
36  * descriptors. This means that a file descriptor can be replaced by
37  * another file descriptor, e.g., standard output can be redirected to
38  * an open file. When the Redirector object is destroyed, the original
39  * file descriptor is restored.
40  *
41  * @see @c dup2 system call.
42  */
43 class Redirector
44   : boost::noncopyable
45 {
46 public:
47   /**
48    * @brief Redirect file descriptor.
49    *
50    * Constructor that sets up file redirection so that future file
51    * access to the file descriptor @e new_file instead operates on the
52    * file given by file descriptor @e old_file.  Internally, before
53    * replacing @e new_file with a copy of @e old_file, a copy of @e
54    * new_file is created, so that the file descriptor @e new_file can
55    * be reset to its original file when the Redirector object is
56    * destroyed.
57    *
58    * The following example demonstrates how this class can be used to
59    * redirect output written to standard output to a logfile.
60    *
61    * @code
62 File logfile( "log.txt", O_WRONLY|O_CREAT, 0666 );
```

```
Redirector redirect( logfile.file(), STDOUT_FILENO );
std::cout << "This appears in log.txt." << std::endl;
@endcode
    *
    * @throws RedirectorError when the redirection cannot be set up.
    */
  Redirector( int old_file, int new_file );
  /**
    * @brief Release redirection.
    *
    * Destructor that releases the redirection and restores the
    * original file associated with the file descriptor @e new_file
    * given when originally constructing the object.
    */
  ~Redirector();

private:
  void restore();

private:
  int new_file_;
  int pre_file_;
};

#endif //DICON_REDIRECT_HPP_
```

Listing B.43: src/redirect.cpp

```
/*--- [Distribution] ------------------------------------------------------
 * This file is part of the Disease Control System DiCon.
 *
 * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
 * Designed and developed with the guidance of Nedialko B. Dimitrov
 * and Lauren Ancel Meyers at the University of Texas at Austin.
 *
 * DiCon is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * DiCon is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
 *-------------------------------------------------------------------------*/

#include "redirect.hpp"
#include "error.hpp"


Redirector::Redirector( int old_file, int new_file )
  : new_file_(new_file)
{
  if( (pre_file_ = dup(new_file_)) == -1 ) {
    BOOST_THROW_EXCEPTION( RedirectorError() << errinfo_api_function("dup")
                           << errinfo_errno(errno) << errinfo_file_descriptor(new_file_) );
  }

  try {
    if( dup2(old_file, new_file_) == -1 ) {
      BOOST_THROW_EXCEPTION( RedirectorError() << errinfo_api_function("dup2") << errinfo_errno(errno)
                             << errinfo_file_descriptor(old_file) << errinfo_file_descriptor_2(new_file_) );
    }
  }
  catch( ... ) {
    restore();
    throw;
  }
}


Redirector::~Redirector() {
  restore();
}


void
Redirector::restore() {
  // Restore previous file desc.
  dup2( pre_file_, new_file_ );
  // Remove backup file desc.
  close( pre_file_ );
}
```

Listing B.44: src/request.hpp

```
#ifndef DICON_REQUEST_HPP_
#define DICON_REQUEST_HPP_

/*--- [Distribution] ------------------------------------------------------
 * This file is part of the Disease Control System DiCon.
 *
 * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
 * Designed and developed with the guidance of Nedialko B. Dimitrov
 * and Lauren Ancel Meyers at the University of Texas at Austin.
 *
 * DiCon is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * DiCon is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
 *-------------------------------------------------------------------------*/

/**
```

```
 * @file
 * @brief RequestQueue class.
 */
#include <boost/noncopyable.hpp>
#include <deque>
#include <map>
#include <vector>


namespace boost {
  namespace mpi {
    class request;
    class status;
  }
}


/**
 * @brief MPI request queue.
 *
 * The templated RequestQueue class implements a queue for MPI
 * requests returned by the Boost MPI library (@c boost::mpi::request
 * class). Each request can be assigned an object of custom type given
 * by the template argument @e T. The queue can be used to wait for
 * any request on it to finish.
 *
 * The queue keeps track of which node each request originated
 * from. This way, requests are finished in the exact order they have
 * been pushed onto the queue, i.e., requests pushed onto the queue
 * later will be returned by wait() only after every other request
 * originating from the same node has been returned. Requests from
 * different nodes can be returned in arbitrary order.
 */
template< typename T >
class RequestQueue
  : boost::noncopyable
{
public:
  /**
   * @brief Push request onto queue.
   *
   * Push the request given by @e request, originating from @e node,
   * to the queue. The user-defined @e data will be returned by wait()
   * when the request finishes.
   *
   * @param node Node the request originates from.
   * @param request MPI request as returned by Boost MPI library.
   * @param data User-defined data to be returned by wait() later.
   */
  void push( int node, const boost::mpi::request &request, const T &data );
  /**
   * @brief Wait for request.
   *
   * Wait for any request on the request queue to finish. This returns
   * the resulting status object of an arbitrary request from the
   * request queue that has already finished, or block until such a
   * request is available. The only limitation to this is that
   * requests from one node will finish in the exact order they have
   * been pushed onto the queue with push(). Requests from different
   * nodes are allowed to finish in arbitrary order.
   *
   * @returns Pair of status object and user-defined object associated
   *   with the request when push() was called.
   *
   * @throws AssertionError when request queue is empty.
   */
  std::pair<boost::mpi::status, T> wait();
  /**
   * @brief Get size of queue.
   *
   * Get the size of the request queue, i.e., the number of pending
   * requests that have not finished and returned a status object
   * through wait().
   *
   * @returns Number of pending requests on queue.
   */
  unsigned size() const;
  /**
   * @brief Check if queue is empty.
   *
   * Check if the request queue is empty. This will return @c false if
   * and only if there is at least one more request to be collected
   * with wait().
   *
   * @returns @c true iff request queue is empty.
   */
  bool empty() const;

private:
  void assert_invariants() const;

private:
  std::vector<boost::mpi::request> front_requests_;
  std::vector<T> front_data_;

private:
  typedef std::map<int, std::deque<std::pair<boost::mpi::request, T> > > back_requests_t;
  back_requests_t back_requests_;
};


#include "request.ipp"

#endif //DICON_REQUEST_HPP_
```

Listing B.45: src/request.ipp

```
// -*- c++ -*-

/*---- [Distribution] ------------------------------------------------
 * This file is part of the Disease Control System DiCon.
 *
 * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
 * Designed and developed with the guidance of Nedialko B. Dimitrov
 * and Lauren Ancel Meyers at the University of Texas at Austin.
 *
 * DiCon is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
```

```
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * DiCon is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
 *----------------------------------------------------------------*/

#include "error.hpp"
#include <boost/foreach.hpp>
#include <boost/mpi/nonblocking.hpp>


template< typename T >
inline
void
RequestQueue<T>::assert_invariants() const {
  assert( !front_requests_.empty() || back_requests_.empty() );
  assert( front_requests_.size() == front_data_.size() );
}


template< typename T >
inline
void
RequestQueue<T>::push( int node, const boost::mpi::request &request, const T &data ) {
  assert_invariants();

  typename back_requests_t::iterator it;

  if( (it = back_requests_.find(node)) != back_requests_.end() ) {
    // We have at least one request for this node in queue.
    it->second.push_back( std::make_pair(request, data) );
  }
  else {
    // There is no request for this node in queue.
    front_requests_.push_back( request );
    front_data_.push_back( data );

    // Create "marker" in back_requests.
    back_requests_[node];
  }

  assert_invariants();
}


template< typename T >
inline
std::pair<boost::mpi::status, T>
RequestQueue<T>::wait() {
  assert_invariants();

  if( front_requests_.empty() )
    BOOST_THROW_EXCEPTION( AssertionError() );

  std::pair<boost::mpi::status, std::vector<boost::mpi::request>::iterator> res
    = boost::mpi::wait_any( front_requests_.begin(), front_requests_.end() );

  const std::vector<boost::mpi::request>::size_type index
    = res.second - front_requests_.begin();
  assert( 0 <= index && index < front_requests_.size() );
  T data = front_data_[index];

  front_requests_[index] = front_requests_.back();
  front_requests_.pop_back();

  front_data_[index] = front_data_.back();
  front_data_.pop_back();

  typename back_requests_t::iterator it
    = back_requests_.find( res.first.source() );

  assert( it != back_requests_.end() );
  if( !it->second.empty() ) {
    // There is at least one more request for this node in queue.
    front_requests_.push_back( it->second.front().first );
    front_data_.push_back( it->second.front().second );
    it->second.pop_front();
  }
  else {
    // No more requests for this node available in queue.
    back_requests_.erase( it );
  }

  assert_invariants();

  return std::make_pair( res.first, data );
}


template< typename T >
inline
unsigned
RequestQueue<T>::size() const {
  assert_invariants();

  unsigned size = front_requests_.size();
  BOOST_FOREACH( const typename back_requests_t::value_type &slot, back_requests_ )
    size += slot.second.size();

  return size;
}


template< typename T >
inline
bool
RequestQueue<T>::empty() const {
  assert_invariants();

  return front_requests_.empty();
}
```

Listing B.46: src/simulator.hpp

```
#ifndef DICON_SIMULATOR_HPP_
#define DICON_SIMULATOR_HPP_

/*---- [Distribution] ------------------------------------------------
 * This file is part of the Disease Control System DiCon.
 *
 * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
 * Designed and developed with the guidance of Nedialko B. Dimitrov
 * and Lauren Ancel Meyers at the University of Texas at Austin.
 *
 * DiCon is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * DiCon is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
 *--------------------------------------------------------------------*/

/**
 * @file
 * @brief Simulator class.
 */
#include "error.hpp"
#include "message.hpp"
#include "../protobuf/cpp/simulator.pb.h"
#include "slavedriver.hpp"


/// %Simulator related error.
struct SimulatorError : virtual Error
{ virtual const char *what() const throw() { return "Simulator related
  error."; } };


/// Calling simulator function failed.
struct SimulatorSlaveError : virtual SimulatorError
{ virtual const char *what() const throw() { return "Calling simulator
  function failed."; } };

/// Calling simulator's children() failed.
struct SimulatorChildrenError : virtual SimulatorSlaveError
{ virtual const char *what() const throw() { return "Calling simulator
  's children() failed."; } };

/// Calling simulator's simulate() failed.
struct SimulatorSimulateError : virtual SimulatorSlaveError
{ virtual const char *what() const throw() { return "Calling simulator
  's simulate() failed."; } };

/// Calling simulator's display() failed.
struct SimulatorDisplayError : virtual SimulatorSlaveError
{ virtual const char *what() const throw() { return "Calling simulator
  's display() failed."; } };


/// %Simulator returned invalid reward value.
struct SimulatorInvalidRewardError : virtual SimulatorSimulateError
{ virtual const char *what() const throw() { return "Simulator
  returned invalid reward value."; } };


/**
 * @brief %Simulator wrapper.
 *
 * The Simulator class wraps a simulator program. It provides the
 * necessary mechanisms for using the Google Protocol Buffer
 * communication layer between the system and the simulator program;
 * and also provides recovery mechanisms in case the simulator program
 * unexpectedly dies.  In this case, two additional attempts will be
 * made per request to restart the simulator program, before an
 * exception is signaled by either of children(), simulate(), or
 * display().
 *
 * The logfile that is given when constructing a Simulator object is
 * never overwritten.  In fact, a new filename is generated whenever
 * the simulator program has to be restarted (in case it unexpectedly
 * died).  A new filename that does not exist is generated by the
 * means of File::unique() each time this happens, using the original
 * filename given on construction as a template.
 *
 * A simulator to be used by the system should implement the following
 * functions and export them through the Google Protocol Buffer
 * communication layer as defined by <code>simulator.proto</code> in
 * the @c protobuf directory distributed with the system. Wrappers
 * for some programming languages are predefined in the @c simulator
 * directory.
 *
 * - <code>%children()</code>: see children().
 * - <code>%simulate()</code>: see simulate().
 * - <code>%display()</code>: see display().
 */
class Simulator
  : public Slavedriver
{
public:
  /**
   * @brief Start simulator.
   *
   * Constructor that runs and initializes the simulator program given
   * by @e command with command-line arguments @e arguments. This
   * also sets up redirection of the simulator's output to standard
   * error, to the logfile given by @e logfile, unless @e logfile is
   * an empty path.  Standard output (not error) of the simulator
   * program is not redirected as this is used for the communication
   * with the system using the Google Protocol Buffers.
   *
   * If the file given by @e logfile already exists, an alternative
   * filename that does not exist will be chosen by the means of
   * File::unique(), so that the original logfile will not be
   * overwritten.
   */
  Simulator( const boost::filesystem::path &logfile
           , const std::string &command, const std::vector<std::string
  > &arguments
```

```
114             );
115
116   public:
117     /**
118      * @brief Get children of policy node.
119      *
120      * Get the children of the given policy node. The policy node in
121      * question is defined by the path leading to it from the root of
122      * the policy tree. Each element in this path must have been
123      * returned by the children() method on a prior invocation. The
124      * empty path is used to get the children at the root of the policy
125      * tree.
126      *
127      * The following code demonstrates how the entire policy space
128      * defined by a simulator can be traversed and displayed in a
129      * reverse depth-first order.
130      *
131      * @code
132   Simulator simulator( ... );
133
134   typedef std::vector<std::string> path_t;
135
136   // Paths still left to do.
137   std::stack<path_t> paths;
138   paths.push( path_t() );
139
140   while( !paths.empty() ) {
141     path_t path = paths.top();
142     paths.pop();
143
144     const std::vector<std::string> &children = simulator.children( path
      );
145
146     if( children.empty() )
147       std::cout << "Leaf: " << simulator.display( path ) << std::endl;
148     else {
149       // Add new paths (path plus child) to path list to do.
150       BOOST_FOREACH( const std::string &child, children ) {
151         path_t new_path = path;
152         new_path.push_back( child );
153         paths.push( new_path );
154       }
155     }
156   }
157   @endcode
158      *
159      * @param path Path to node in question.
160      * @returns List of children at that node.
161      *
162      * @throws SimulatorChildrenError when getting list of children from
163      *   simulator (<code>%children()</code>) failed.
164      */
165     std::vector<std::string> children( const std::vector<std::string> &
      path );
166     /**
167      * @brief Simulate the given policy.
168      *
169      * Simulate the given policy and return the reward value. The policy
170      * must be a path in the policy tree where a prior call to
171      * children() returned an empty list. If the reward value returned
172      * from the simulator program is less than 0 or greater than 1, this
173      * call fails.
174      *
175      * @param policy Policy to simulate.
176      * @returns Reward value for policy.
177      *
178      * @throws SimulatorSimulateError when simulating the policy in
179      *   simulator (<code>%simulate()</code>) failed.
180      * @throws SimulatorInvalidRewardError when the reward value was
181      *   less than 0 or greater than 1.
182      */
183     double simulate( const policy_t &policy );
184     /**
185      * @brief Display the given policy.
186      *
187      * Display the policy given by @e policy. This is expected to return
188      * a human-readable interpretation of the given policy. It is up to
189      * the simulator program to decide what is an appropriate
190      * interpretation of the given policy.
191      *
192      * @param policy Policy to display.
193      * @returns Human-readable interpretation.
194      *
195      * @throws SimulatorDisplayError when displaying the policy in the
196      *   simulator (<code>%display()</code>) failed.
197      */
198     std::string display( const policy_t &policy );
199
200   private:
201     void process( const proto::simulator::Question &question, proto::
      simulator::Answer &answer );
202   };
203
204   #endif //DICON_SIMULATOR_HPP_
```

Listing B.47: src/simulator.cpp

```
1   /*---- [Distribution] ------------------------------------------------
2    * This file is part of the Disease Control System DiCon.
3    *
4    * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
5    * Designed and developed with the guidance of Nedialko B. Dimitrov
6    * and Lauren Ancel Meyers at the University of Texas at Austin.
7    *
8    * DiCon is free software: you can redistribute it and/or modify it
9    * under the terms of the GNU General Public License as published by
10   * the Free Software Foundation, either version 3 of the License, or
11   * (at your option) any later version.
12   *
13   * DiCon is distributed in the hope that it will be useful, but
14   * WITHOUT ANY WARRANTY; without even the implied warranty of
15   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
16   * General Public License for more details.
17   *
18   * You should have received a copy of the GNU General Public License
19   * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
20   *--------------------------------------------------------------------*/
21
22  #include "simulator.hpp"
23  #include "optimizer.hpp"
```

```
24 #include <boost/foreach.hpp>
25
26
27 Simulator::Simulator( const boost::filesystem::path &logfile
28                       , const std::string &command, const std::vector<std::string> &arguments
29                       )
30   : Slavedriver(logfile, command, arguments)
31 {
32 }
33
34
35 void
36 Simulator::process( const proto::simulator::Question &question, proto::simulator::Answer &answer ) {
37   execute( question, answer );
38
39   if( answer.failed() )
40     BOOST_THROW_EXCEPTION( SimulatorSlaveError() << errinfo_slave_error(answer.a_failure().what()) );
41 }
42
43
44 std::vector<std::string>
45 Simulator::children( const std::vector<std::string> &path ) {
46   try {
47     proto::simulator::Question question;
48
49     // Formulate question.
50
51     question.mutable_q_children();
52     BOOST_FOREACH( const std::string &node, path )
53       question.mutable_q_children()->add_node( node );
54
55     question.set_type( proto::simulator::CHILDREN );
56
57     // Parse answer.
58
59     proto::simulator::Answer answer;
60     process( question, answer );
61
62     std::vector<std::string> children;
63     BOOST_FOREACH( const std::string &node, answer.a_children().node() )
64       children.push_back( node );
65
66     return children;
67   }
68   catch( ... ) {
69     BOOST_THROW_EXCEPTION( SimulatorChildrenError()
70                            << errinfo_nested_exception(boost::current_exception()) );
71   }
72 }
73
74
75 double
76 Simulator::simulate( const policy_t &policy ) {
77   try {
78     proto::simulator::Question question;
79
80     // Formulate question.
81
82     question.mutable_q_simulate();
83     BOOST_FOREACH( const std::string &node, policy )
84       question.mutable_q_simulate()->add_node( node );
85
86     question.set_type( proto::simulator::SIMULATE );
87
88     // Parse answer.
89
90     proto::simulator::Answer answer;
91     process( question, answer );
92
93     // Clip result.
94
95     double reward = answer.a_simulate().reward();
96
97     if( !std::isfinite(reward) )
98       BOOST_THROW_EXCEPTION( SimulatorInvalidRewardError() << errinfo_value<double>(reward) );
99
100    double clipped = reward < 0 ? 0 : (reward > 1 ? 1 : reward);
101
102    if( fabs(reward - clipped) > 1e-6 ) {
103      BOOST_THROW_EXCEPTION( SimulatorInvalidRewardError() << errinfo_max_value<double>(1)
104                             << errinfo_value<double>(reward) << errinfo_min_value<double>(0) );
105    }
106
107    return clipped;
108  }
109  catch( SimulatorInvalidRewardError & ) {
110    // Do not catch with "..." below.
111    throw;
112  }
113  catch( ... ) {
114    BOOST_THROW_EXCEPTION( SimulatorSimulateError()
115                           << errinfo_nested_exception(boost::current_exception()) );
116  }
117 }
118
119
120 std::string
121 Simulator::display( const policy_t &policy ) {
122   try {
123     proto::simulator::Question question;
124
125     // Formulate question.
126
127     question.mutable_q_display();
128     BOOST_FOREACH( const std::string &node, policy )
129       question.mutable_q_display()->add_node( node );
130
131     question.set_type( proto::simulator::DISPLAY );
132
133     // Parse answer.
134
135     proto::simulator::Answer answer;
136     process( question, answer );
```

```
137
138       return answer.a_display().display();
139     }
140     catch( ... ) {
141       BOOST_THROW_EXCEPTION( SimulatorDisplayError()
142                              << errinfo_nested_exception(boost::
    current_exception()) );
143     }
144 }
```

Listing B.48: src/slavedriver.hpp

```
1  #ifndef DICON_SLAVEDRIVER_HPP_
2  #define DICON_SLAVEDRIVER_HPP_
3
4  /*---- [Distribution] ------------------------------------------------
5   * This file is part of the Disease Control System DiCon.
6   *
7   * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8   * Designed and developed with the guidance of Nedialko B. Dimitrov
9   * and Lauren Ancel Meyers at the University of Texas at Austin.
10  *
11  * DiCon is free software: you can redistribute it and/or modify it
12  * under the terms of the GNU General Public License as published by
13  * the Free Software Foundation, either version 3 of the License, or
14  * (at your option) any later version.
15  *
16  * DiCon is distributed in the hope that it will be useful, but
17  * WITHOUT ANY WARRANTY; without even the implied warranty of
18  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19  * General Public License for more details.
20  *
21  * You should have received a copy of the GNU General Public License
22  * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23  *--------------------------------------------------------------------*/
24
25 /**
26  * @file
27  * @brief Slavedriver class.
28  */
29 #include "file.hpp"
30 #include <boost/filesystem/path.hpp>
31 #include <boost/noncopyable.hpp>
32 #include <boost/optional.hpp>
33 #include <boost/scoped_ptr.hpp>
34 #include <string>
35 #include <vector>
36
37
38 /// Errinfo storing the child's process ID.
39 DICON_ERRINFO( slave_pid, pid_t );
40
41
42 /// %Slavedriver related error.
43 struct SlavedriverError : virtual Error
44 { virtual const char *what() const throw() { return "Slavedriver_
    related_error."; } };
45
46 /// Failed to execute given query.
47 struct SlavedriverExecuteError : virtual SlavedriverError
48 { virtual const char *what() const throw() { return "Failed_to_execute
    _given_query."; } };
49
50
51 class Communicator;
52
53 /**
54  * @brief Running child process.
55  *
56  * The Slavedriver class implements running a child process. It offers
57  * methods to communicate with the process using standard input and
58  * output using the simple %message protocol described in the
59  * documentation of the Communicator class. If the child process dies
60  * unexpectedly during a query, i.e., without returning a full
61  * response, it is automatically restarted until communication
62  * succeeds or the given retry limit is reached.
63  *
64  * In addition, the child's output to standard error can be redirected
65  * to a logfile. If the logfile given on construction already exists,
66  * it will not be overwritten, but instead a new, unused filename will
67  * be chosen by the means of File::unique(). This will be repeated
68  * whenever the child process has to be restarted (using the original
69  * filename given on construction as a template), so that old logfiles
70  * from previous runs are never lost.
71  */
72 class Slavedriver
73   : boost::noncopyable
74 {
75 public:
76   /// @showinitializer
77   /// Default number of retries on execute().
78   static const unsigned default_retries = 2;
79
80 public:
81   /**
82    * @brief Create slavedriver.
83    *
84    * Constructor that creates a new slavedriver. This does not spawn
85    * the child process until execute() is called.
86    *
87    * @param logfile Logfile to used for child's standard error output.
88    * @param command Command of child process to execute.
89    * @param arguments Command-line arguments.
90    */
91   Slavedriver( const boost::filesystem::path &logfile,
92                const std::string &command, const std::vector<std::
    string> &arguments );
93   /**
94    * @brief Kill slavedriver.
95    *
96    * Destructor that destroy the slavedriver and kills any child
97    * process that is still running. When killing a child process, the
98    * slavedriver will first send the hang-up signal (SIGHUP) to the
99    * process. If it still exists after 5 seconds, the kill signal
100   * (SIGKILL) will be sent, ultimately killing the process.
101   */
102  ~Slavedriver();
103
104 public:
105   /**
```

```
106     * @brief Reset child process.
107     *
108     * Reset the child process. If a child process is currently running,
109     * it will be killed as described in ~Slavedriver(). If a child
110     * process does not exist, this call does nothing.
111     */
112    void reset();
113    /**
114     * @brief Reset child process with new arguments.
115     *
116     * Reset the child process, changing the command-line arguments. If
117     * a child process is currently running, it will be killed as
118     * described in ~Slavedriver(). When the child process is next
119     * started, the new command-line arguments will be used.
120     *
121     * @param arguments New command-line arguments.
122     */
123    void reset( const std::vector<std::string> &arguments );
124    /**
125     * @brief Reset child process with new command and arguments.
126     *
127     * Reset the child process, changing both the command to execute and
128     * the command-line arguments. If a child process is currently
129     * running, it will be killed as described in ~Slavedriver(). When
130     * the child process is next started, the new command and
131     * command-line arguments will be used.
132     *
133     * @param command New command of child process to execute.
134     * @param arguments New command-line arguments.
135     */
136    void reset( const std::string &command, const std::vector<std::
    string> &arguments );
137
138  protected:
139    /**
140     * @brief Run query on child.
141     *
142     * Run the given query on the child process, using the communication
143     * protocol described in the Communicator class, on the child's
144     * standard input and standard output. If a child process is not
145     * running, it will be started with the command and command-line
146     * arguments specified in the constructor or as given at the last
147     * call of either @link reset(const std::vector<std::string> &)
    reset(vector<string>)@endlink
148     * or @link reset(const std::string &, const std::vector<std::string
    > &) reset(string, vector<string>)@endlink.
149     *
150     * In case the child process dies unexpectedly before sending a
151     * complete answer to the query, it will be automatically restarted
152     * up to @e retries times.
153     *
154     * @param query Query to send.
155     * @param retries Number of retries.
156     * @returns Child's answer to given query.
157     *
158     * @throws SlavedriverExecuteError when querying child ultimately
159     *   failed.
160     */
161    std::string execute( const std::string &query, unsigned retries =
    default_retries );
162
163    /**
164     * @brief Run query on child.
165     *
166     * Run the given query on the child process. This method is
167     * equivalent to the non-templated execute() method, except that it
168     * uses the Google Protocol Buffer library to serialize the object
169     * @e question and then sends this serialized version of the object
170     * to the child process. Similarly, the child's answer is
171     * deserialized using the Google Protocol Buffer library and
172     * returned in @e answer.
173     *
174     * @note This uses the @e SerializeToOstream(std::ostream*) method
175     *   on @e question, and the @e ParseFromIstream(std::istream*)
176     *   method on @e answer. Any objects that can provide these two
177     *   methods can be used here.
178     */
179    template< typename Q, typename A >
180    void execute( const Q &question, A &answer, unsigned retries =
    default_retries );
181
182  private:
183    boost::optional<File> logfile();
184
185    void check_child();
186    void kill_child();
187    void run_child();
188
189  private:
190    boost::filesystem::path logfile_;
191
192    mutable pid_t child_;
193
194    std::string command_;
195    std::vector<std::string> arguments_;
196
197    boost::scoped_ptr<Communicator> communicator_;
198  };
199
200
201  #include "slavedriver.ipp"
202
203  #endif //DICON_SLAVEDRIVER_HPP_
```

Listing B.49: src/slavedriver.ipp

```
1   // -*- c++ -*-
2
3   /*---- [Distribution] ------------------------------------------------
4    * This file is part of the Disease Control System DiCon.
5    *
6    * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
7    * Designed and developed with the guidance of Nedialko B. Dimitrov
8    * and Lauren Ancel Meyers at the University of Texas at Austin.
9    *
10   * DiCon is free software: you can redistribute it and/or modify it
11   * under the terms of the GNU General Public License as published by
12   * the Free Software Foundation, either version 3 of the License, or
13   * (at your option) any later version.
14   *
```

```
 * DiCon is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
 *-------------------------------------------------------------------*/

#include <sstream>


template< typename Q, typename A >
void
Slavedriver::execute( const Q &question, A &answer, unsigned retries )
{
  std::stringstream buffer;

  question.SerializeToOstream( &buffer );
  buffer.str( execute(buffer.str(), retries) );
  answer.ParseFromIstream( &buffer );
}
```

Listing B.50: src/slavedriver.cpp

```
/*--- [Distribution] ------------------------------------------------*
 * This file is part of the Disease Control System DiCon.
 *
 * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
 * Designed and developed with the guidance of Nedialko B. Dimitrov
 * and Lauren Ancel Meyers at the University of Texas at Austin.
 *
 * DiCon is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * DiCon is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
 *-------------------------------------------------------------------*/

#include "slavedriver.hpp"
#include "communicator.hpp"
#include "error.hpp"
#include "pipe.hpp"
#include <boost/filesystem/operations.hpp>
#include <boost/format.hpp>
#include <cerrno>
#include <iostream>
#include <sys/wait.h>


Slavedriver::Slavedriver( const boost::filesystem::path &logfile,
                          const std::string &command, const std::vector<std::string> &arguments )
  : logfile_(logfile), child_(0), command_(command), arguments_(arguments)
{
}


Slavedriver::~Slavedriver() {
  try {
    kill_child();
  }
  catch( ... ) {
    // Ignore.
  }
}


void
Slavedriver::reset() {
  reset( command_, arguments_ );
}


void
Slavedriver::reset( const std::vector<std::string> &arguments ) {
  reset( command_, arguments );
}


void
Slavedriver::reset( const std::string &command, const std::vector<std::string> &arguments ) {
  kill_child();

  command_ = command;
  arguments_ = arguments;
}


std::string
Slavedriver::execute( const std::string &query, unsigned retries ) {
  while( true ) {
    try {
      run_child();
      assert( communicator_ );
      communicator_->send_message( query );
      return communicator_->receive_message();
    }
    catch( boost::exception &e ) {
      if( retries-- != 0 ) {
        kill_child();
        sleep( 1 );
        continue;
      }
      else {
        BOOST_THROW_EXCEPTION( SlavedriverExecuteError()
                               << errinfo_nested_exception(boost::copy_exception(e)) );
      }
    }
```

```
91    }
92  }
93
94
95  boost::optional<File>
96  Slavedriver::logfile() {
97    if( logfile_.empty() )
98      return boost::none;
99    else
100     return File::unique( logfile_ );
101 }
102
103
104 void
105 Slavedriver::check_child() {
106   if( child_ == 0 ) {
107     // Nothing to do.
108     return;
109   }
110
111   pid_t res;
112   if( (res = waitpid(child_, NULL, WNOHANG)) == -1 ) {
113     BOOST_THROW_EXCEPTION( SystemError() << errinfo_api_function("waitpid")
114                            << errinfo_errno(errno) << errinfo_slave_pid(child_) );
115   }
116
117   assert( res == 0 || res == child_ );
118
119   if( res == child_ ) {
120     // Child is dead now.
121     child_ = 0;
122     communicator_.reset();
123   }
124 }
125
126
127 void
128 Slavedriver::kill_child() {
129   if( child_ == 0 ) {
130     // Nothing to do.
131     return;
132   }
133
134   // Send the hang-up signal.
135   if( kill(child_, SIGHUP) != 0 ) {
136     BOOST_THROW_EXCEPTION( SystemError() << errinfo_api_function("kill")
137                            << errinfo_errno(errno) << errinfo_slave_pid(child_) );
138   }
139
140   // Give child process five seconds to shutdown gracefully.
141   for( unsigned i = 0; child_ != 0 && i < 5000; ++i ) {
142     usleep( 1000 );
143     check_child();
144   }
145
146   if( child_ == 0 ) {
147     // Nothing to do.
148     return;
149   }
150
151   // Send the kill signal.
152   if( kill(child_, SIGKILL) != 0 ) {
153     BOOST_THROW_EXCEPTION( SystemError() << errinfo_api_function("kill")
154                            << errinfo_errno(errno) << errinfo_slave_pid(child_) );
155   }
156
157   if( waitpid(child_, NULL, 0) != child_ ) {
158     BOOST_THROW_EXCEPTION( SystemError() << errinfo_api_function("waitpid")
159                            << errinfo_errno(errno) << errinfo_slave_pid(child_) );
160   }
161
162   child_ = 0;
163   communicator_.reset();
164 }
165
166
167
168 namespace detail {
169
170   void
171   dup2( int oldfd, int newfd ) {
172     if( ::dup2(oldfd, newfd) == -1 ) {
173       BOOST_THROW_EXCEPTION( SystemError() << errinfo_api_function("dup2") << errinfo_errno(errno)
174                              << errinfo_file_descriptor(oldfd) << errinfo_file_descriptor_2(newfd) );
175     }
176   }
177
178 }
179
180
181
182 void
183 Slavedriver::run_child() {
184   check_child();
185
186   if( child_ != 0 ) {
187     // Nothing to do.
188     return;
189   }
190
191   // Create pipes.
192
193   Pipe pipe_pc;  // parent -> child
194   Pipe pipe_cp;  // child -> parent
195
196   // Create error logfile.
197
198   boost::optional<File> logfile = this->logfile();
199
200   // Prepare exec() arguments.
201
202   const char *path;
```

```
203     std::vector<char *> argv( arguments_.size()+2 );
204
205     path = argv[0] = const_cast<char *>(command_.c_str());
206
207     for( unsigned i = 0; i < arguments_.size(); ++i )
208       argv[i+1] = const_cast<char *>(arguments_[i].c_str());
209
210     argv[arguments_.size()+1] = NULL;
211
212     // Fork child process.
213
214     pid_t child;
215
216     switch( (child = fork()) ) {
217     case -1:
218       BOOST_THROW_EXCEPTION( SystemError() << errinfo_api_function("fork
      ") << errinfo_errno(errno) );
219
220     case 0:
221       // Child process.
222       close( pipe_cp.read_end(true) );  // read end of child -> parent
223       close( pipe_pc.write_end(true) );  // write end of parent -> child
224
225       try {
226         try {
227           detail::dup2( pipe_pc.read_end(), STDIN_FILENO );
228           close( pipe_pc.read_end(true) );
229
230           detail::dup2( pipe_cp.write_end(), STDOUT_FILENO );
231           close( pipe_cp.write_end(true) );
232
233           if( logfile ) {
234             detail::dup2( logfile->file(), STDERR_FILENO );
235             close( logfile->file(true) );
236           }
237
238           execv( path, argv.data() );
239
240           // If we reach this, something went wrong.
241           BOOST_THROW_EXCEPTION( SystemError() << errinfo_api_function("
      execv")
242                                  << errinfo_errno(errno) << 
      errinfo_value<std::string>(path) );
243         }
244 #if BOOST_VERSION >= 103900
245         catch( ... ) {
246           std::cerr << "Caught_unhandled_exception_in_fork'd_child_
      process.\n"
247                     << boost::current_exception_diagnostic_information()
      ;
248           std::cerr << std::flush;
249         }
250 #else
251         catch( boost::exception &e ) {
252           std::cerr << "Caught_unhandled_exception_in_fork'd_child_
      process.\n"
253                     << boost::diagnostic_information( e );
254           std::cerr << std::flush;
255         }
256         catch( ... ) {
257           std::cerr << "Caught_unhandled_exception_in_fork'd_child_
      process.\n"
258                     << "<unknown_exception>\n";
259           std::cerr << std::flush;
260         }
261 #endif
262       }
263       catch( ... ) {
264         // Ignore all exceptions. We have to call exit and not let
265         // exceptions fall through here because we don't want any
266         // destructors to be executed inside the child process.
267       }
268
269       exit( EXIT_FAILURE );
270
271     default:
272       // Parent process.
273       close( pipe_pc.read_end(true) );  // read end of parent -> child
274       close( pipe_cp.write_end(true) );  // write end of child -> parent
275
276       if( logfile )
277         close( logfile->file(true) );
278
279       communicator_.reset( new Communicator(pipe_cp.read_end(), pipe_pc.
      write_end()) );
280       pipe_pc.write_end( true );
281       pipe_cp.read_end( true );
282       child_ = child;
283     }
284 }
```

Listing B.51: src/specifier.hpp

```
1  #ifndef DICON_SPECIFIER_HPP_
2  #define DICON_SPECIFIER_HPP_
3
4  /*--- [Distribution] ---------------------------------------------------
5   * This file is part of the Disease Control System DiCon.
6   *
7   * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8   * Designed and developed with the guidance of Nedialko B. Dimitrov
9   * and Lauren Ancel Meyers at the University of Texas at Austin.
10  *
11  * DiCon is free software: you can redistribute it and/or modify it
12  * under the terms of the GNU General Public License as published by
13  * the Free Software Foundation, either version 3 of the License, or
14  * (at your option) any later version.
15  *
16  * DiCon is distributed in the hope that it will be useful, but
17  * WITHOUT ANY WARRANTY; without even the implied warranty of
18  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19  * General Public License for more details.
20  *
21  * You should have received a copy of the GNU General Public License
22  * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23  *-----------------------------------------------------------------------*/
24
25 /**
26  * @file
```

```
 * @brief Specifier parser.
 *
 * The specifier.hpp file provides the parser for argument and
 * schedule specifiers such as used in the main configuration
 * parameters @c sim_args and @c opt_orgs, and @c checkpoints. The
 * complete grammar for both argument specifiers
 * (<code>ArgumentSpecifier</code>) and schedule specifiers
 * (<code>ScheduleSpecifier</code>) is given in Extended Backus–Naur
 * Form (EBNF, as per ISO/IEC 14977:1996(E)) as follows.
 *
 * @verbatim
ArgumentSpecifier = Set ;
ScheduleSpecifier = "[" , ( RegularRange | ExpressionRange ) , "]"
                  | Expression , { "," , Expression } ;

Set = Specifier , { "|" , Specifier } ;
Specifier = SpecifierElement , { SpecifierElement } ;
SpecifierElement = "[" , ( RegularRange | ExpressionRange ) , "]"
                 | "{" , Set , "}"
                 | Literal ;

RegularRange = RegularRangeElement , { "|" , RegularRangeElement } ;
RegularRangeElement = Singleton | SimpleRange | ExtendedRange ;

ExpressionRange = Expression , ":" , ( ExpressionSubrange , { ":" ,
↪ ExpressionSubrange } ) ;
ExpressionSubrange = [ Identifier , "=" ] , RegularRange ;

Singleton = Expression ;
SimpleRange = Expression , ".." , [ Expression ] ;
ExtendedRange = Expression , "," , Expression , ".." , [ Expression ]
↪ ;

Expression = Term , { ( "+" | "-" ) , Term } ;
Term = Factor , { ( "*" | "/" ) , Factor } ;
Factor = Group , "^" , Factor
       | [ "+" | "-" ] , Group ;
Group = "(" , Expression , ")"
      | FunctionCall | Identifier | Real ;
FunctionCall = FunctionName , "(" , Expression , ")" ;
Identifier = ( Letter | "_" ) , { Letter | Digit | "_" } ;

Literal = LiteralElement , { LiteralElement } ;
LiteralElement = "'" , SingleQuoted , "'"
               | '"' , DoubleQuoted , '"'
               | RegularCharacter ;
SingleQuoted = ? any character except ' ? ;
DoubleQuoted = ? any character except " ? ;
RegularCharacter = ? any character except ', ", |, [, ], {, } ? ;

Real = [ "+" | "-" ] , Digits , [ "." , Digits ] , [ ( "e" | "E" ) , [
↪ "+" | "-" ] , Digits ] ;
Digits = Digit , { Digit } ;

Digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
Letter = ? any character either in range "a" to "z", or "A" to "Z" ? ;
FunctionName = "sqrt"  | "round"  | "trunc" | "floor" | "ceil"  | "abs
↪ "
             | "log"   | "log2"   | "log10" | "exp"   | "exp2"  | "
↪ exp10"
             | "cos"   | "sin"    | "tan"   | "acos"  | "asin"  | "
↪ atan"
             | "cosh"  | "sinh"   | "tanh"  | "acosh" | "asinh" | "
↪ atanh"
             | "erf"   | "erfc"   | "j0"    | "j1"    | "y0"    | "y1"
             | "gamma" | "lgamma" | "log1p" | "expm1" ;
@endverbatim
 *
 * The following are examples for argument specifiers
 * (<code>ArgumentSpecifier</code>) with the sets of arguments they
 * represent.
 *
 * @verbatim
foo|bar                          -> {foo, bar}
[1..10]                          -> {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
[0,0.1..1]                       -> {0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6,
↪ 0.7, 0.8, 0.9, 1}
[1..10|10,20..90]                -> {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20,
↪ 30, 40, 50, 60, 70, 80, 90}
[x^2+y^2:x=1..6:y=2..4]          -> {5, 10, 17, 8, 13, 20, 13, 18, 25,
↪ 20, 25, 32, 29, 34, 41, 40, 45, 52}
--foo=[1..2]|--bar=[3..9]        -> {--foo=1, --foo=2, --bar=3, --bar
↪ =4, --bar=5, --bar=6, --bar=7, --bar=8, --bar=9}
--method={a|b} --value=[1..2]  -> {--method=a --value=1, --method=a
↪ --value=2, --method=b --value=1, --method=b --value=2}
@endverbatim
 *
 * The following are examples for schedule specifiers
 * (<code>ScheduleSpecifier</code>) with the sets of numbers they
 * represent.
 *
 * @verbatim
1,4,9,2,6,10   -> {1, 2, 4, 6, 9, 10}
[2..4|9,7..1]  -> {1, 2, 3, 4, 5, 7, 9}
[2^(x*2):0..]  -> {1, 4, 16, 64, 256, ...}
@endverbatim
 *
 * @see http://en.wikipedia.org/wiki/Extended_Backus–Naur_Form
 */
#include "error.hpp"
#include "lazy/set.hpp"
#include "types.hpp"
#include <string>


/// Errinfo storing function name in specifier.
DICON_ERRINFO( specifier_function_name, std::string );
/// Errinfo storing variable name in specifier.
DICON_ERRINFO( specifier_variable_name, std::string );


/// Specifier related error.
struct SpecifierError : virtual Error
{ virtual const char *what() const throw() { return "Specifier_related
↪ _error."; } };

/// Failed to parse specifier.
struct SpecifierParseError : virtual SpecifierError
{ virtual const char *what() const throw() { return "Failed_to_parse_
↪ specifier."; } };

```

```
134 /// Failed to parse specifier range.
135 struct SpecifierRangeError : virtual SpecifierParseError
136 { virtual const char *what() const throw() { return "Failed to parse
    specifier range."; } };
137
138 /// Range in specifier has zero step size.
139 struct SpecifierRangeZeroStepError : virtual SpecifierRangeError
140 { virtual const char *what() const throw() { return "Range in
    specifier has zero step size."; } };
141
142 /// Unbounded decreasing range in specifier.
143 struct SpecifierRangeUnboundedError : virtual SpecifierRangeError
144 { virtual const char *what() const throw() { return "Unbounded
    decreasing range in specifier."; } };
145
146 /// Variable not defined in specifier.
147 struct SpecifierUndefinedVariableError : virtual SpecifierParseError
148 { virtual const char *what() const throw() { return "Variable not
    defined in specifier."; } };
149
150 /// Function not defined in specifier.
151 struct SpecifierUndefinedFunctionError : virtual SpecifierParseError
152 { virtual const char *what() const throw() { return "Function not
    defined in specifier."; } };
153
154 /// Variable already defined in specifier.
155 struct SpecifierVariableRedefinedError : virtual SpecifierParseError
156 { virtual const char *what() const throw() { return "Variable already
    defined in specifier."; } };
157
158 /// Too many variables in specifier.
159 struct SpecifierTooManyVariablesError : virtual SpecifierParseError
160 { virtual const char *what() const throw() { return "Too many
    variables in specifier."; } };
161
162 /// Variable not used in specifier.
163 struct SpecifierUnusedVariableError : virtual SpecifierParseError
164 { virtual const char *what() const throw() { return "Variable not used
    in specifier."; } };
165
166 /// Invalid simulation count produced by specifier.
167 struct SpecifierInvalidSimcountError : virtual SpecifierError
168 { virtual const char *what() const throw() { return "Invalid
    simulation count produced by specifier."; } };
169
170
171 /**
172  * @brief Parse argument specifier.
173  *
174  * Parse the string given by @e specifier, representing an argument
175  * specifier. The complete grammar for such a specifier is given in
176  * the documentation of the specifier.hpp file.
177  *
178  * This call returns a lazy set representing the arguments specified
179  * by the given string.
180  *
181  * @param specifier String representing argument specifier.
182  * @returns Lazy set containing the arguments specified.
183  *
184  * @throws SpecifierParseError or a derived class when the given
185  *   specifier cannot be parsed.
186  */
187 LazySet<std::string>::ptr_t parse_argument_specifier( const std::
    string &specifier );
188 /**
189  * @brief Parse schedule specifier.
190  *
191  * Parse the string given by @e specifier, representing a schedule
192  * specifier. The complete grammar for such a specifier is given in
193  * the documentation of the specifier.hpp file.
194  *
195  * This call returns a lazy set representing the schedule, i.e., the
196  * list of simulation counts, specified by the given string.
197  *
198  * The elements in the set returned by this function are of type @link
199  * ::simcount_t simcount_t@endlink. When an element in the set is
200  * requested (LazySet::get()) that does not represent a simulation
201  * count, i.e., it is either not a positive integer or cannot be
202  * represented in the range defined by the @link ::simcount_t
203  * simcount_t@endlink type, an exception of type
204  * SpecifierInvalidSimcountError is thrown. As the set represented by
205  * the given specifier can be infinite, this check can only be done
206  * when the invalid element in question has been reached.
207  * Specifically, this check cannot be done at the time the specifier
208  * is parsed.
209  *
210  * @param specifier String representing schedule specifier.
211  * @returns Lazy set containing the schedule specified.
212  *
213  * @throws SpecifierParseError or a derived class when the given
214  *   specifier cannot be parsed.
215  */
216 LazySet<simcount_t>::ptr_t parse_schedule_specifier( const std::string
    &specifier );
217
218 #endif //DICON_SPECIFIER_HPP_
```

Listing B.52: src/specifier.cpp

```
1  /*---- [Distribution] ------------------------------------------------
2   * This file is part of the Disease Control System DiCon.
3   *
4   * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
5   * Designed and developed with the guidance of Nedialko B. Dimitrov
6   * and Lauren Ancel Meyers at the University of Texas at Austin.
7   *
8   * DiCon is free software: you can redistribute it and/or modify it
9   * under the terms of the GNU General Public License as published by
10  * the Free Software Foundation, either version 3 of the License, or
11  * (at your option) any later version.
12  *
13  * DiCon is distributed in the hope that it will be useful, but
14  * WITHOUT ANY WARRANTY; without even the implied warranty of
15  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
16  * General Public License for more details.
17  *
18  * You should have received a copy of the GNU General Public License
19  * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
20  *--------------------------------------------------------------------*/
21
```

```
22 #include "specifier.hpp"
23 #include "config.hpp"
24 #include "lazy/set/combine.hpp"
25 #include "lazy/set/compose.hpp"
26 #include "lazy/set/concat.hpp"
27 #include "lazy/set/container.hpp"
28 #include "lazy/set/range.hpp"
29 #include "lazy/set/singleton.hpp"
30 #include "lazy/set/transform.hpp"
31 #include "lazy/set/union.hpp"
32 #include "lazy/value/binary.hpp"
33 #include "lazy/value/constant.hpp"
34 #include "lazy/value/unary.hpp"
35 #include "lazy/value/variable.hpp"
36 #include <boost/assign/list_of.hpp>
37 #include <boost/foreach.hpp>
38 #include <boost/function.hpp>
39 #include <boost/lambda/lambda.hpp>
40 #include <boost/lexical_cast.hpp>
41 #include <boost/regex.hpp>
42 #include <boost/spirit/include/classic.hpp>
43 #include <boost/spirit/include/classic_grammar_def.hpp>
44 #include <boost/spirit/include/classic_parse_tree.hpp>
45 #include <boost/version.hpp>
46 #include <cmath>
47 #include <gmpxx.h>
48
49
50 #define DICON_ASSERT_PARSER_ID_(WHAT, ID)                                \
   \
51   do { if( (WHAT) != SpecifierGrammar::ID )                              \
   \
52        BOOST_THROW_EXCEPTION( AssertionError()                           \
   \
53                               << errinfo_value<parser_id>((WHAT))        \
   \
54                               << errinfo_expected_value<parser_id>       \
   \
55                               (SpecifierGrammar::ID) );                  \
   \
56   } while( false )                                                       \
   \
57     /**/
58
59 #define DICON_ASSERT_SIZE_(WHAT, VALUE)                                  \
   \
60   do { if( !((WHAT) == (VALUE)) )                                        \
   \
61        BOOST_THROW_EXCEPTION( AssertionError()                           \
   \
62                               << errinfo_value<size_t>((WHAT))           \
   \
63                               << errinfo_expected_value<size_t>          \
   \
64                               ((VALUE)) );                               \
   \
65   } while( false )                                                       \
   \
66     /**/
67
```

```
68 #define DICON_ASSERT_SIZE_MIN_(WHAT, VALUE)                              \
   \
69   do { if( !((WHAT) >= (VALUE)) )                                        \
   \
70        BOOST_THROW_EXCEPTION( AssertionError()                           \
   \
71                               << errinfo_value<size_t>((WHAT))           \
   \
72                               << errinfo_min_value<size_t>               \
   \
73                               ((VALUE)) );                               \
   \
74   } while( false )                                                       \
   \
75     /**/
76
77
78 namespace detail {
79
80   typedef double real_t;
81   typedef mpq_class rational_t;
82
83
84   // We choose 'real_to_rational_prec' deliberately slightly larger
85   // than 'real_to_string_prec' for the following reason. When using
86   // expression ranges with additional expressions in the (sub)range
87   // specifiers themselves, we want intuitive behavior, e.g., in the
88   // following situation:
89   //
90   //    [x*10+y*3:x=1..10:y=(x)|(x+1/3)|(x+2/3)]
91   //
92   // The range should return only integer values. If both precisions
93   // were the same, this would not necessarily happen: range bounds
94   // are first rounded with 'real_to_rational_prec', and then the
95   // result of the calculating (which thus only has an effective
96   // precision of 14), is rounded again.
97   //
98   // In order to conceal this lowered precision, we again round the
99   // result, but with even slightly lower precision. Obviously, this
100  // is a compromise between having "strange" artifacts (range bounds
101  // missing in range) and precision of calculations. For the purpose
102  // of floating-point arithmetic operations, 12 significant digits
103  // are probably acceptable.
104
105  static const std::streamsize real_to_string_prec = 12;
106  static const std::streamsize real_to_rational_prec = 14;
107
108
109  static const std::map<std::string, double(*)(double)>
   exp_range_functions
110   = boost::assign::map_list_of<std::string, double(*)(double)>
111 #define DICON_NAME_FUNCTION_( x ) (#x, static_cast<double(*)(double)
   >(::x))
112     DICON_NAME_FUNCTION_( sqrt   )
113     DICON_NAME_FUNCTION_( round  )
114     DICON_NAME_FUNCTION_( trunc  )
115     DICON_NAME_FUNCTION_( floor  )
116     DICON_NAME_FUNCTION_( ceil   )
117     (             "abs", ::fabs  )
118
119     DICON_NAME_FUNCTION_( log    )
```

```
    DICON_NAME_FUNCTION_( log2    )
    DICON_NAME_FUNCTION_( log10   )

    DICON_NAME_FUNCTION_( exp     )
    DICON_NAME_FUNCTION_( exp2    )
    DICON_NAME_FUNCTION_( exp10   )

    DICON_NAME_FUNCTION_( cos     )
    DICON_NAME_FUNCTION_( sin     )
    DICON_NAME_FUNCTION_( tan     )

    DICON_NAME_FUNCTION_( acos    )
    DICON_NAME_FUNCTION_( asin    )
    DICON_NAME_FUNCTION_( atan    )

    DICON_NAME_FUNCTION_( cosh    )
    DICON_NAME_FUNCTION_( sinh    )
    DICON_NAME_FUNCTION_( tanh    )

    DICON_NAME_FUNCTION_( acosh   )
    DICON_NAME_FUNCTION_( asinh   )
    DICON_NAME_FUNCTION_( atanh   )

    (             "gamma", ::tgamma )
    (            "lgamma", ::lgamma )

    DICON_NAME_FUNCTION_( log1p   )
    DICON_NAME_FUNCTION_( expm1   )

    DICON_NAME_FUNCTION_( erf     )
    DICON_NAME_FUNCTION_( erfc    )

    DICON_NAME_FUNCTION_( j0      )
    DICON_NAME_FUNCTION_( j1      )

    DICON_NAME_FUNCTION_( y0      )
    DICON_NAME_FUNCTION_( y1      )
#undef DICON_NAME_FUNCTION_
    ;


  using namespace boost::spirit::classic;

  struct SpecifierGrammar
    : public grammar<SpecifierGrammar>
  {
    enum
    { specifier_id = 1

    , set_id
    , range_id
    , exp_range_id
    , exp_subrange_id

    , lazy_range_id
    , lazy_singleton_id
    , lazy_simple_range_id
    , lazy_extended_range_id
    , exp_or_real_id

    , expression_id
    , term_id
    , factor_id
    , group_id

    , identifier_id
    , literal_id
    , real_id

    , argument_id
    , schedule_id
    };

    enum
    { argument = 0
    , schedule = 1
    };

    template< typename scanner_t >
    struct definition
      : grammar_def< rule<scanner_t, parser_tag<argument_id> >
                   , rule<scanner_t, parser_tag<schedule_id> >
                   >
    {
      definition( const SpecifierGrammar &self ) {
        specifier          = +( (no_node_d[ch_p('{')] >> (set
      ) >> no_node_d[ch_p('}')])
                              | (no_node_d[ch_p('[')] >> (exp_range |
 range) >> no_node_d[ch_p(']')])
                              | (                          (literal
      )                        ) );

        set                =  specifier % no_node_d[ch_p('|')];
        range              =  lazy_range;
        exp_range          =  expression
                           >> no_node_d[ch_p(':')]
                           >> (exp_subrange % no_node_d[ch_p(':')]);
        exp_subrange       =  !(identifier >> no_node_d[ch_p('=')])
                           >> lazy_range;

        lazy_range         =  (lazy_extended_range |
 lazy_simple_range | lazy_singleton) % no_node_d[ch_p('|')];
        lazy_singleton     =  exp_or_real;
        lazy_simple_range  =  exp_or_real
                           >> no_node_d[str_p("..")] >> !exp_or_real;
        lazy_extended_range =  exp_or_real
                           >> no_node_d[ch_p (',' )] >>  exp_or_real
                           >> no_node_d[str_p("..")] >> !exp_or_real;
        exp_or_real        =  (expression - real) | real;

        expression         =  term % chset_p("+-");
        term               =  factor % chset_p("*/");
        factor             =  ( (group >> ch_p('^') >> factor)
                              | (chset_p("+-") >> group)
                              | group
                              );
        group              =  ( (identifier >> no_node_d[ch_p('(')]
 >> expression >> no_node_d[ch_p(')')])
                              | (                 no_node_d[ch_p('(')]
 >> expression >> no_node_d[ch_p(')')])
                              | identifier
                              | real
```

```
                                );

        identifier          =  leaf_node_d[   (alpha_p | ch_p('_')
        )
                                        >> *(alpha_p | ch_p('_') | digit_p)
                                        ];
        literal             =  leaf_node_d[  +( (ch_p('"') >> *(~ch_p('"')) >> ch_p('"'))
                                          | (ch_p('\'') >> *(~ch_p('\'')) >> ch_p('\''))
                                          | +(~chset_p("|[]{}'\"") )
                                          ) ];
        real                =  leaf_node_d[  !chset_p("+-")
                                        >> +digit_p
                                        >> !(ch_p    ('.')  >>        +digit_p)
                                        >> !(chset_p("eE") >> !chset_p("+-") >> +digit_p)
                                        ];

        argument            =  set;
        schedule            =  ( (no_node_d[ch_p('[')] >> (exp_range | range) >> no_node_d[ch_p(']')])
                                | (exp_or_real % no_node_d[ch_p(',')])
                                );

        start_parsers( argument, schedule );
      }

      rule<scanner_t, parser_tag<specifier_id          > > specifier;

      rule<scanner_t, parser_tag<set_id                > > set;
      rule<scanner_t, parser_tag<range_id              > > range;
      rule<scanner_t, parser_tag<exp_range_id          > > exp_range;
      rule<scanner_t, parser_tag<exp_subrange_id       > > exp_subrange;

      rule<scanner_t, parser_tag<lazy_range_id         > > lazy_range;
      rule<scanner_t, parser_tag<lazy_singleton_id     > > lazy_singleton;
      rule<scanner_t, parser_tag<lazy_simple_range_id  > > lazy_simple_range;
      rule<scanner_t, parser_tag<lazy_extended_range_id> > lazy_extended_range;
      rule<scanner_t, parser_tag<exp_or_real_id        > > exp_or_real;

      rule<scanner_t, parser_tag<expression_id         > > expression;
      rule<scanner_t, parser_tag<term_id               > > term;
      rule<scanner_t, parser_tag<factor_id             > > factor;
      rule<scanner_t, parser_tag<group_id              > > group;

      rule<scanner_t, parser_tag<identifier_id         > > identifier;
      rule<scanner_t, parser_tag<literal_id            > > literal;
      rule<scanner_t, parser_tag<real_id               > > real;

      rule<scanner_t, parser_tag<argument_id           > > argument;
      rule<scanner_t, parser_tag<schedule_id           > > schedule;
    };
  };


  class ExpressionOrReal {
  public:
    ExpressionOrReal( LazyValue<real_t>::ptr_t &expression )
      : expression_(expression)
    {
    }

    ExpressionOrReal( const rational_t &real )
      : real_(real)
    {
    }

    bool is_expression() const { return expression_; }
    bool is_real() const { return !is_expression(); }

    const LazyValue<real_t> &expression() const {
      assert( is_expression() );
      return *expression_;
    }

    const rational_t &real() const {
      assert( is_real() );
      return real_;
    }

  private:
    boost::shared_ptr<LazyValue<real_t> > expression_;
    rational_t real_;
  };


  class LazyRange
    : boost::noncopyable
  {
  public:
    typedef rational_t value_t;
    typedef std::auto_ptr<LazyRange> ptr_t;
    typedef std::set<std::string> references_t;
    typedef std::map<std::string, real_t> variables_t;

  public:
    virtual ~LazyRange() {}
    virtual references_t references() const = 0;
    virtual LazySet<rational_t>::ptr_t operator()( const variables_t &variables ) const = 0;
  };


  static
  rational_t
  string_to_rational( const std::string &x ) {
#if BOOST_VERSION >= 104000
    boost::regex regex("(?<sgn>[+-])?(?<int>\\d+)(?:\\.(?<dec>\\d+))?(?:[eE](?<exp>[+-]?\\d+))?");
#else
    boost::regex regex("(    "[+-])?("    "\\d+)(?:\\."    "\\d+))?(?:[eE]("    "[+-]?\\d+))?");
#endif
```

```
342
343       boost::smatch match;
344       if( !boost::regex_match(x, match, regex) )
345         BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<std::
    string>(x) );
346
347       assert( match.size() == 5 );
348 #if BOOST_VERSION >= 104000
349       const std::string &sgn_part = match.str("sgn");
350       const std::string &int_part = match.str("int");
351       const std::string &dec_part = match.str("dec");
352       const std::string &exp_part = match.str("exp");
353 #else
354       const std::string &sgn_part = match.str(1);
355       const std::string &int_part = match.str(2);
356       const std::string &dec_part = match.str(3);
357       const std::string &exp_part = match.str(4);
358 #endif
359
360       rational_t res;
361       res += rational_t(int_part, 10);
362
363       if( !dec_part.empty() )
364         res += rational_t(dec_part, 10) / rational_t("1" + std::string(
    dec_part.length(), '0'), 10);
365
366       if( !exp_part.empty() ) {
367         ssize_t exp = boost::lexical_cast<ssize_t>( exp_part );
368
369         if( exp > 0 )
370           res *= rational_t("1" + std::string( exp, '0'), 10);
371         else if( exp < 0 )
372           res /= rational_t("1" + std::string(-exp, '0'), 10);
373       }
374
375       if( sgn_part == "-" )
376         res *= -1;
377
378       return res;
379     }
380
381
382     struct RealToString {
383       typedef std::string result_type;
384       typedef real_t argument_type;
385
386       RealToString( std::streamsize prec = real_to_string_prec )
387         : prec(prec)
388       {
389       }
390
391       result_type
392       operator()( const argument_type &x ) const {
393         if( !std::isfinite(x) )
394           BOOST_THROW_EXCEPTION( CalculationNotFiniteError() <<
    errinfo_value<argument_type>(x) );
395
396         std::stringstream str;
397         str.precision( prec );
398         str << mpf_class(x);
399         return str.str();
400       }
401
402       std::streamsize prec;
403     };
404
405
406     struct RationalToString {
407       typedef std::string result_type;
408       typedef rational_t argument_type;
409
410       result_type
411       operator()( const argument_type &x ) const {
412         std::stringstream str;
413         str.precision( 12 );
414         str << mpf_class(x);
415         return str.str();
416       }
417     };
418
419
420     struct RealToRational {
421       typedef rational_t result_type;
422       typedef real_t argument_type;
423
424       result_type
425       operator()( const argument_type &x ) const {
426         // This adds a slight rounding (14 significant digits where
    real_t
427         // can provide roughly 16 digits). This makes for more
    predictable
428         // behavior, as on some system, e.g., the expression
    "7/10*7*10/7"
429         // evaluates to something slightly less than 7. If this was used
430         // as an upper bound in an interval this upper bound would not
431         // be included in this interval, quite counterintuitively.
432
433         return string_to_rational( RealToString(real_to_rational_prec)(x
    ) );
434       }
435     };
436
437
438     class RangeFactory
439       : public LazyRange
440     {
441     public:
442       RangeFactory( const ExpressionOrReal &value )
443         : lower_(value), sort_(false), open_(false)
444       {
445       }
446
447       RangeFactory( const ExpressionOrReal &lower, const boost::optional
    <ExpressionOrReal> &upper, bool sort )
448         : lower_(lower), upper_(upper), sort_(sort), open_(!upper)
449       {
450       }
451
452       RangeFactory( const ExpressionOrReal &lower, const
    ExpressionOrReal &first, const boost::optional<ExpressionOrReal> &
    upper, bool sort )
```

```
453       : lower_(lower), first_(first), upper_(upper), sort_(sort), ↩
  ↪ open_(!upper)
454     {
455     }
456
457   private:
458     static
459     void
460     add_to_set( references_t &references, const boost::optional<↩
  ↪ ExpressionOrReal> &x ) {
461       if( x && x->is_expression() ) {
462         const references_t &refs = x->expression().references();
463         references.insert( refs.begin(), refs.end() );
464       }
465     }
466
467     static
468     rational_t
469     evaluate( const boost::optional<ExpressionOrReal> &x, const ↩
  ↪ variables_t &variables, const rational_t &fallback ) {
470       if( x )
471         return x->is_expression() ? RealToRational()(x->expression()(↩
  ↪ variables)) : x->real();
472       else
473         return fallback;
474     }
475
476     static
477     rational_t
478     trunc_rational( const rational_t &x ) {
479       return rational_t( x.get_num() / x.get_den() );
480     }
481
482     static
483     LazySet<rational_t>::ptr_t
484     create_range( const rational_t &lower, const rational_t &step, ↩
  ↪ bool sort ) {
485       if( step == 0 ) {
486         BOOST_THROW_EXCEPTION( SpecifierRangeZeroStepError()
487                                << errinfo_value<std::string>((boost::↩
  ↪ format("[%s,%s..]")
488                                                               % ↩
  ↪ RationalToString()(lower)
489                                                               % ↩
  ↪ RationalToString()(lower+step)).str()) );
490       }
491
492       if( !sort || step >= 0 )
493         return lazy_range( lower, step );
494       else {
495         BOOST_THROW_EXCEPTION( SpecifierRangeUnboundedError()
496                                << errinfo_value<std::string>((boost::↩
  ↪ format("[%s,%s..]")
497                                                               % ↩
  ↪ RationalToString()(lower)
498                                                               % ↩
  ↪ RationalToString()(lower+step)).str()) );
499       }
500     }
501
502     static
503     LazySet<rational_t>::ptr_t
504     create_range( const rational_t &lower, const rational_t &upper, ↩
  ↪ const rational_t &step, bool sort ) {
505       if( step == 0 ) {
506         BOOST_THROW_EXCEPTION( SpecifierRangeZeroStepError()
507                                << errinfo_value<std::string>((boost::↩
  ↪ format("[%s,%s..%s]")
508                                                               % ↩
  ↪ RationalToString()(lower)
509                                                               % ↩
  ↪ RationalToString()(lower+step)
510                                                               % ↩
  ↪ RationalToString()(upper)).str()) );
511       }
512
513       if( !sort || step >= 0 || upper > lower )
514         return lazy_range( lower, upper, step );
515       else {
516         // In order for merging (with lazy_union) to work, range must
517         // be ordered. The following code exactly reverses the range.
518
519         return lazy_range( rational_t(lower + trunc_rational((upper - ↩
  ↪ lower) / step) * step), lower, rational_t(-step) );
520       }
521     }
522
523   public:
524     references_t
525     references() const {
526       references_t references;
527
528       add_to_set( references, lower_ );
529       add_to_set( references, first_ );
530       add_to_set( references, upper_ );
531
532       return references;
533     }
534
535     LazySet<rational_t>::ptr_t
536     operator()( const variables_t &variables ) const {
537       rational_t lower = evaluate( lower_, variables, 0       );
538       rational_t upper = evaluate( upper_, variables, lower   );
539       rational_t first = evaluate( first_, variables, lower+1 );
540
541       if( open_ )
542         return create_range( lower,       rational_t(first-lower), ↩
  ↪ sort_ );
543       else
544         return create_range( lower, upper, rational_t(first-lower), ↩
  ↪ sort_ );
545     }
546
547   private:
548     boost::optional<ExpressionOrReal> lower_;
549     boost::optional<ExpressionOrReal> first_;
550     boost::optional<ExpressionOrReal> upper_;
551     bool sort_;
552     bool open_;
553   };
554
555   static
```

```
556   LazyRange::ptr_t
557   range_factory( const ExpressionOrReal &value ) {
558     return LazyRange::ptr_t( new RangeFactory(value) );
559   }
560
561   static
562   LazyRange::ptr_t
563   range_factory( const ExpressionOrReal &lower,
564                  const boost::optional<ExpressionOrReal> &upper, bool sort )
565   {
566     return LazyRange::ptr_t( new RangeFactory(lower, upper, sort) );
567   }
568
569   static
570   LazyRange::ptr_t
571   range_factory( const ExpressionOrReal &lower,
572                  const ExpressionOrReal &first,
573                  const boost::optional<ExpressionOrReal> &upper, bool sort )
574   {
575     return LazyRange::ptr_t( new RangeFactory(lower, first, upper, sort) );
576   }
577
578
579   class RangeUnion
580     : public LazyRange
581   {
582   public:
583     RangeUnion( LazyRange::ptr_t &a, LazyRange::ptr_t &b )
584       : a_(a), b_(b)
585     {
586     }
587
588   private:
589     static
590     void
591     add_to_set( references_t &references, const LazyRange::ptr_t &x ) {
592       const references_t &refs = x->references();
593       references.insert( refs.begin(), refs.end() );
594     }
595
596   public:
597     virtual
598     references_t
599     references() const {
600       references_t references;
601
602       add_to_set( references, a_ );
603       add_to_set( references, b_ );
604
605       return references;
606     }
607
608     virtual
609     LazySet<rational_t>::ptr_t
610     operator()( const variables_t &variables ) const {
611       return lazy_union( (*a_)(variables), (*b_)(variables) );
612     }
613
614   private:
615     const LazyRange::ptr_t a_;
616     const LazyRange::ptr_t b_;
617   };
618
619   static
620   LazyRange::ptr_t
621   range_union( LazyRange::ptr_t &a, LazyRange::ptr_t &b ) {
622     return LazyRange::ptr_t( new RangeUnion(a, b) );
623   }
624
625
626   static LazySet<std::string>::ptr_t parse_specifier          ( const tree_node<node_val_data<>> &node                    );
627
628   static LazySet<std::string>::ptr_t parse_set                ( const tree_node<node_val_data<>> &node                    );
629   static LazySet<rational_t>::ptr_t  parse_range              ( const tree_node<node_val_data<>> &node, bool sort = false );
630   static LazySet<real_t>::ptr_t      parse_exp_range          ( const tree_node<node_val_data<>> &node                    );
631   static LazyRange::ptr_t            parse_exp_subrange       ( const tree_node<node_val_data<>> &node, std::string &name );
632
633   static LazyRange::ptr_t            parse_lazy_range         ( const tree_node<node_val_data<>> &node, bool sort = false );
634   static LazyRange::ptr_t            parse_lazy_singleton     ( const tree_node<node_val_data<>> &node                    );
635   static LazyRange::ptr_t            parse_lazy_simple_range  ( const tree_node<node_val_data<>> &node, bool sort         );
636   static LazyRange::ptr_t            parse_lazy_extended_range( const tree_node<node_val_data<>> &node, bool sort         );
637   static ExpressionOrReal            parse_exp_or_real        ( const tree_node<node_val_data<>> &node                    );
638
639   static LazyValue<real_t>::ptr_t    parse_expression         ( const tree_node<node_val_data<>> &node                    );
640   static LazyValue<real_t>::ptr_t    parse_term               ( const tree_node<node_val_data<>> &node                    );
641   static LazyValue<real_t>::ptr_t    parse_factor             ( const tree_node<node_val_data<>> &node                    );
642   static LazyValue<real_t>::ptr_t    parse_group              ( const tree_node<node_val_data<>> &node                    );
643
644   static std::string                 parse_identifier         ( const tree_node<node_val_data<>> &node                    );
645   static std::string                 parse_literal            ( const tree_node<node_val_data<>> &node                    );
646   static rational_t                  parse_real               ( const tree_node<node_val_data<>> &node                    );
647
648   static LazySet<std::string>::ptr_t parse_argument           ( const tree_node<node_val_data<>> &node                    );
649   static LazySet<simcount_t>::ptr_t  parse_schedule           ( const tree_node<node_val_data<>> &node                    );
650
651
652   static
653   LazySet<std::string>::ptr_t
654   parse_specifier( const tree_node<node_val_data<>> &node ) {
```

```
655     DICON_ASSERT_PARSER_ID_( node.value.id(), specifier_id );
656     DICON_ASSERT_SIZE_MIN_( node.children.size(), 1 );
657
658     LazySet<std::string>::ptr_t combined;
659
660     BOOST_FOREACH( const tree_node<node_val_data<> > &child, node.children ) {
661       LazySet<std::string>::ptr_t current;
662
663       if( child.value.id() == SpecifierGrammar::set_id )
664         current = parse_set( child );
665       else if( child.value.id() == SpecifierGrammar::exp_range_id )
666         current = lazy_transform( parse_exp_range(child), RealToString() );
667       else if( child.value.id() == SpecifierGrammar::range_id )
668         current = lazy_transform( parse_range(child), RationalToString() );
669       else if( child.value.id() == SpecifierGrammar::literal_id )
670         current = lazy_singleton( parse_literal(child) );
671       else
672         BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<parser_id>(child.value.id()) );
673
674       assert( current.get() );
675       if( combined.get() ) {
676         combined = lazy_combine( combined, current,
677                                  std::plus<std::string>() );
678       }
679       else
680         combined = current;
681     }
682
683     assert( combined.get() );
684     return combined;
685   }
686
687
688   static
689   LazySet<std::string>::ptr_t
690   parse_set( const tree_node<node_val_data<> > &node ) {
691     DICON_ASSERT_PARSER_ID_( node.value.id(), set_id );
692     DICON_ASSERT_SIZE_MIN_( node.children.size(), 1 );
693
694     LazySet<std::string>::ptr_t combined;
695
696     BOOST_FOREACH( const tree_node<node_val_data<> > &child, node.children ) {
697       LazySet<std::string>::ptr_t current;
698
699       if( child.value.id() == SpecifierGrammar::specifier_id )
700         current = parse_specifier( child );
701       else
702         BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<parser_id>(child.value.id()) );
703
704       assert( current.get() );
705       if( combined.get() )
706         combined = lazy_concat( combined, current );
707       else
708         combined = current;
709     }
710
711     assert( combined.get() );
712     return combined;
713   }
714
715
716   static
717   LazySet<rational_t>::ptr_t
718   parse_range( const tree_node<node_val_data<> > &node, bool sort ) {
719     DICON_ASSERT_PARSER_ID_( node.value.id(), range_id );
720     DICON_ASSERT_SIZE_( node.children.size(), 1 );
721
722     LazyRange::ptr_t range = parse_lazy_range( node.children.front(), sort );
723     const LazyRange::references_t &references = range->references();
724
725     if( !references.empty() )
726       BOOST_THROW_EXCEPTION( SpecifierUndefinedVariableError() << errinfo_specifier_variable_name(*references.begin()) );
727
728     return (*range)( LazyRange::variables_t() );
729   }
730
731
732   template< typename C, typename T = typename C::value_type >
733   struct Insert {
734     typedef C result_type;
735     typedef C first_argument_type;
736     typedef T second_argument_type;
737
738     result_type
739     operator()( first_argument_type x, const second_argument_type &y ) const {
740       x.insert( y );
741       return x;
742     }
743   };
744
745
746   struct RationalToReal {
747     typedef real_t result_type;
748     typedef rational_t argument_type;
749
750     result_type
751     operator()( const argument_type &x ) const {
752       return x.get_d();
753     }
754   };
755
756
757   template< typename T1, typename T2 >
758   struct MakePair2nd {
759     typedef typename std::pair<T1, T2> result_type;
760     typedef T2 argument_type;
761
762     MakePair2nd( const T1 &first )
763       : first(first)
764     {
765     }
766
767     result_type
```

```
768     operator()( const argument_type &second ) const {
769       return std::make_pair<T1, T2>( first, second );
770     }
771
772     const T1 first;
773   };
774
775
776   struct PushRange {
777     typedef LazySet<std::map<std::string, real_t> >::ptr_t result_type;
778     typedef std::map<std::string, real_t> argument_type;
779
780     PushRange( const std::string &name, LazyRange::ptr_t &range )
781       : name(name), range(range)
782     {
783     }
784
785     result_type
786     operator()( const argument_type &variables ) const {
787       LazySet<rational_t>::ptr_t range = (*this->range)( variables );
788
789       return lazy_combine( lazy_singleton(variables)
790                          , lazy_transform( lazy_transform(range, RationalToReal()),
791                                            MakePair2nd<std::string, real_t>(name) )
792                          , Insert<std::map<std::string, real_t> >()
793                          );
794     }
795
796     const std::string name;
797     const boost::shared_ptr<LazyRange> range;
798   };
799
800
801   static
802   std::string
803   first_range_name() {
804     return "x";
805   }
806
807
808   static
809   void
810   next_range_name( std::string &name ) {
811     assert( name.length() == 1 );
812
813     if( 'x' <= name[0] && name[0] < 'z' )
814       ++name[0];
815     else if( name[0] == 'z' )
816       name[0] = 'a';
817     else if( 'a' <= name[0] && name[0] < 'x'-1 )
818       ++name[0];
819     else
820       BOOST_THROW_EXCEPTION( SpecifierTooManyVariablesError() );
821   }
822
823
824   template< typename T >
825   struct ApplyExpression {
826     typedef T result_type;
827     typedef typename LazyValue<T>::variables_t argument_type;
828
829     ApplyExpression( typename LazyValue<T>::ptr_t &expression )
830       : expression(expression)
831     {
832     }
833
834     result_type
835     operator()( const argument_type &variables ) const {
836       return (*expression)( variables );
837     }
838
839     const boost::shared_ptr<LazyValue<T> > expression;
840   };
841
842
843   static
844   LazySet<real_t>::ptr_t
845   parse_exp_range( const tree_node<node_val_data<> > &node ) {
846     DICON_ASSERT_PARSER_ID_( node.value.id(), exp_range_id );
847     DICON_ASSERT_SIZE_MIN_( node.children.size(), 2 );
848
849     std::set<std::string> vars_defined, vars_used;
850     LazySet<std::map<std::string, real_t> >::ptr_t combined = lazy_singleton( std::map<std::string, real_t>() );
851
852     boost::optional<std::string> fallback_name;
853     for( size_t i = 1; i < node.children.size(); ++i ) {
854       std::string name;
855       LazyRange::ptr_t range = parse_exp_subrange( node.children[i], name );
856
857       if( name.empty() ) {
858         if( fallback_name )
859           next_range_name( *fallback_name );
860         else
861           fallback_name = first_range_name();
862
863         name = *fallback_name;
864       }
865
866       if( (vars_defined.find(name)) != vars_defined.end() )
867         BOOST_THROW_EXCEPTION( SpecifierVariableRedefinedError() << errinfo_specifier_variable_name(name) );
868
869       BOOST_FOREACH( const std::string &ref_name, range->references() ) {
870         if( vars_defined.find(ref_name) == vars_defined.end() )
871           BOOST_THROW_EXCEPTION( SpecifierUndefinedVariableError() << errinfo_specifier_variable_name(ref_name) );
872
873         vars_used.insert( ref_name );
874       }
875
876       assert( combined.get() );
877       combined = lazy_compose( combined, PushRange(name, range) );
878
879       vars_defined.insert( name );
880     }
881
```

```
882     LazyValue<real_t>::ptr_t expression = parse_expression( node.children[0] );
883
884     BOOST_FOREACH( const std::string &ref_name, expression->references() ) {
885       if( vars_defined.find(ref_name) == vars_defined.end() )
886         BOOST_THROW_EXCEPTION( SpecifierUndefinedVariableError() << errinfo_specifier_variable_name(ref_name) );
887
888       vars_used.insert( ref_name );
889     }
890
891     BOOST_FOREACH( const std::string &name, vars_defined ) {
892       if( vars_used.find(name) == vars_used.end() )
893         BOOST_THROW_EXCEPTION( SpecifierUnusedVariableError() << errinfo_specifier_variable_name(name) );
894     }
895
896     assert( combined.get() );
897     return lazy_transform( combined, ApplyExpression<real_t>(expression) );
898   }
899
900
901   static
902   LazyRange::ptr_t
903   parse_exp_subrange( const tree_node<node_val_data<> > &node, std::string &name ) {
904     DICON_ASSERT_PARSER_ID_( node.value.id(), exp_subrange_id );
905
906     if( node.children.size() == 2 )
907       name = parse_identifier( node.children[0] );
908     else if( node.children.size() == 1 )
909       name.clear();
910     else
911       BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<size_t>(node.children.size()) );
912
913     return parse_lazy_range( node.children[node.children.size()-1] );
914   }
915
916
917   static
918   LazyRange::ptr_t
919   parse_lazy_range( const tree_node<node_val_data<> > &node, bool sort_ ) {
920     DICON_ASSERT_PARSER_ID_( node.value.id(), lazy_range_id );
921     DICON_ASSERT_SIZE_MIN_( node.children.size(), 1 );
922
923     LazyRange::ptr_t combined;
924
925     const bool sort = sort_ || (node.children.size() > 1);
926     BOOST_FOREACH( const tree_node<node_val_data<> > &child, node.children ) {
927       LazyRange::ptr_t current;
928
929       if( child.value.id() == SpecifierGrammar::lazy_extended_range_id )
930         current = parse_lazy_extended_range( child, sort );
931       else if( child.value.id() == SpecifierGrammar::lazy_simple_range_id )
932         current = parse_lazy_simple_range( child, sort );
933       else if( child.value.id() == SpecifierGrammar::lazy_singleton_id )
934         current = parse_lazy_singleton( child );
935       else
936         BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<parser_id>(child.value.id()) );
937
938       assert( current.get() );
939       if( combined.get() )
940         combined = range_union( combined, current );
941       else
942         combined = current;
943     }
944
945     assert( combined.get() );
946     return combined;
947   }
948
949
950   static
951   LazyRange::ptr_t
952   parse_lazy_singleton( const tree_node<node_val_data<> > &node ) {
953     DICON_ASSERT_PARSER_ID_( node.value.id(), lazy_singleton_id );
954     DICON_ASSERT_SIZE_( node.children.size(), 1 );
955
956     return range_factory( parse_exp_or_real(node.children[0]) );
957   }
958
959
960   static
961   LazyRange::ptr_t
962   parse_lazy_simple_range( const tree_node<node_val_data<> > &node, bool sort ) {
963     DICON_ASSERT_PARSER_ID_( node.value.id(), lazy_simple_range_id );
964
965     if( node.children.size() == 2 ) {
966       return range_factory( parse_exp_or_real(node.children[0]),
967                             parse_exp_or_real(node.children[1]), sort );
968     }
969     else if( node.children.size() == 1 ) {
970       return range_factory( parse_exp_or_real(node.children[0]),
971                             boost::none                        , sort );
972     }
973     else
974       BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<size_t>(node.children.size()) );
975   }
976
977
978   static
979   LazyRange::ptr_t
980   parse_lazy_extended_range( const tree_node<node_val_data<> > &node, bool sort ) {
981     DICON_ASSERT_PARSER_ID_( node.value.id(), lazy_extended_range_id );
982
983     if( node.children.size() == 3 ) {
984       return range_factory( parse_exp_or_real(node.children[0]),
```

```
985                             parse_exp_or_real(node.children[1]),
986                             parse_exp_or_real(node.children[2]), sort );
987       }
988       else if( node.children.size() == 2 ) {
989         return range_factory( parse_exp_or_real(node.children[0]),
990                             parse_exp_or_real(node.children[1]),
991                             boost::none                         , sort );
992       }
993       else
994         BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<size_t>(node.children.size()) );
995     }
996
997
998     static
999     ExpressionOrReal
1000    parse_exp_or_real( const tree_node<node_val_data<> > &node ) {
1001      DICON_ASSERT_PARSER_ID_( node.value.id(), exp_or_real_id );
1002      DICON_ASSERT_SIZE_( node.children.size(), 1 );
1003
1004      const tree_node<node_val_data<> > &child = node.children.front();
1005
1006      if( child.value.id() == SpecifierGrammar::expression_id ) {
1007        LazyValue<real_t>::ptr_t expression = parse_expression( child );
1008        return expression;
1009      }
1010      else if( child.value.id() == SpecifierGrammar::real_id )
1011        return parse_real( child );
1012      else
1013        BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<parser_id>(child.value.id()) );
1014    }
1015
1016
1017    static
1018    LazyValue<real_t>::ptr_t
1019    parse_expression( const tree_node<node_val_data<> > &node ) {
1020      DICON_ASSERT_PARSER_ID_( node.value.id(), expression_id );
1021
1022      if( !(node.children.size() >= 1 && node.children.size() % 2 == 1) )
1023        BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<size_t>(node.children.size()) );
1024
1025      LazyValue<real_t>::ptr_t combined;
1026
1027      for( size_t i = 0; i < node.children.size(); i += 2 ) {
1028        LazyValue<real_t>::ptr_t current = parse_term( node.children[i] );
1029
1030        if( i != 0 ) {
1031          std::string op( node.children[i-1].value.begin(),
1032                          node.children[i-1].value.end() );
1033
1034          boost::function<real_t(real_t, real_t)> function;
1035
1036          if( op == "+" )
1037            function = boost::lambda::_1 + boost::lambda::_2;
1038          else if( op == "-" )
1039            function = boost::lambda::_1 - boost::lambda::_2;
1040          else
1041            BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<std::string>(op) );
1042
1043          assert( combined.get() );
1044          combined = lazy_binary_op( combined, current, function );
1045        }
1046        else
1047          combined = current;
1048      }
1049
1050      assert( combined.get() );
1051      return combined;
1052    }
1053
1054
1055    static
1056    LazyValue<real_t>::ptr_t
1057    parse_term( const tree_node<node_val_data<> > &node ) {
1058      DICON_ASSERT_PARSER_ID_( node.value.id(), term_id );
1059
1060      if( !(node.children.size() >= 1 && node.children.size() % 2 == 1) )
1061        BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<size_t>(node.children.size()) );
1062
1063      LazyValue<real_t>::ptr_t combined;
1064
1065      for( size_t i = 0; i < node.children.size(); i += 2 ) {
1066        LazyValue<real_t>::ptr_t current = parse_factor( node.children[i] );
1067
1068        if( i != 0 ) {
1069          std::string op( node.children[i-1].value.begin(),
1070                          node.children[i-1].value.end() );
1071
1072          boost::function<real_t(real_t, real_t)> function;
1073
1074          if( op == "*" )
1075            function = boost::lambda::_1 * boost::lambda::_2;
1076          else if( op == "/" )
1077            function = boost::lambda::_1 / boost::lambda::_2;
1078          else
1079            BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<std::string>(op) );
1080
1081          assert( combined.get() );
1082          combined = lazy_binary_op( combined, current, function );
1083        }
1084        else
1085          combined = current;
1086      }
1087
1088      assert( combined.get() );
1089      return combined;
1090    }
1091
1092
1093    static
1094    LazyValue<real_t>::ptr_t
```

```
1095    parse_factor( const tree_node<node_val_data<> > &node ) {
1096      DICON_ASSERT_PARSER_ID_( node.value.id(), factor_id );
1097
1098      if( node.children.size() == 3 ) {
1099        DICON_ASSERT_PARSER_ID_( node.children[0].value.id(), group_id );
1100        DICON_ASSERT_PARSER_ID_( node.children[1].value.id(), factor_id );
1101        DICON_ASSERT_PARSER_ID_( node.children[2].value.id(), factor_id );
1102
1103        std::string op( node.children[1].value.begin(),
1104                        node.children[1].value.end() );
1105
1106        boost::function<real_t(real_t, real_t)> function;
1107
1108        if( op == "^" )
1109          function = ::pow;
1110        else
1111          BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<std::string>(op) );
1112
1113        return lazy_binary_op( parse_group(node.children[0]),
1114                               parse_factor(node.children[2]), function );
1115      }
1116      else if( node.children.size() == 2 ) {
1117        DICON_ASSERT_PARSER_ID_( node.children[0].value.id(), factor_id );
1118        DICON_ASSERT_PARSER_ID_( node.children[1].value.id(), group_id );
1119
1120        std::string op( node.children[0].value.begin(),
1121                        node.children[0].value.end() );
1122
1123        boost::function<real_t(real_t)> function;
1124
1125        if( op == "+" )
1126          function = +boost::lambda::_1;
1127        else if( op == "-" )
1128          function = -boost::lambda::_1;
1129        else
1130          BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<std::string>(op) );
1131
1132        return lazy_unary_op( parse_group(node.children[1]), function );
1133      }
1134      else if( node.children.size() == 1 )
1135        return parse_group( node.children.front() );
1136      else
1137        BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<size_t>(node.children.size()) );
1138    }
1139
1140
1141    static
1142    LazyValue<real_t>::ptr_t
1143    parse_group( const tree_node<node_val_data<> > &node ) {
1144      DICON_ASSERT_PARSER_ID_( node.value.id(), group_id );
1145
1146      if( node.children.size() == 2 ) {
1147        DICON_ASSERT_PARSER_ID_( node.children[0].value.id(), identifier_id );
1148        DICON_ASSERT_PARSER_ID_( node.children[1].value.id(), expression_id );
1149
1150        const std::string &name = parse_identifier( node.children[0] );
1151        boost::function<real_t(real_t)> function;
1152
1153        try {
1154          function = exp_range_functions.at( name );
1155        }
1156        catch( std::out_of_range & ) {
1157          BOOST_THROW_EXCEPTION( SpecifierUndefinedFunctionError() << errinfo_specifier_function_name(name) );
1158        }
1159
1160        assert( function );
1161        return lazy_unary_op( parse_expression(node.children[1]), function );
1162      }
1163      else if( node.children.size() == 1 ) {
1164        const tree_node<node_val_data<> > &child = node.children.front();
1165
1166        if( child.value.id() == SpecifierGrammar::expression_id )
1167          return parse_expression( child );
1168        else if( child.value.id() == SpecifierGrammar::identifier_id )
1169          return lazy_variable<real_t>( parse_identifier(child) );
1170        else if( child.value.id() == SpecifierGrammar::real_id )
1171          return lazy_constant( RationalToReal()(parse_real(child)) );
1172        else
1173          BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<parser_id>(child.value.id()) );
1174      }
1175      else
1176        BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<size_t>(node.children.size()) );
1177    }
1178
1179
1180    static
1181    std::string
1182    parse_identifier( const tree_node<node_val_data<> > &node ) {
1183      DICON_ASSERT_PARSER_ID_( node.value.id(), identifier_id );
1184      DICON_ASSERT_SIZE_( node.children.size(), 1 );
1185      DICON_ASSERT_PARSER_ID_( node.children.front().value.id(), identifier_id );
1186
1187      return std::string( node.children.front().value.begin(),
1188                          node.children.front().value.end() );
1189    }
1190
1191
1192    static
1193    std::string
1194    parse_literal( const tree_node<node_val_data<> > &node ) {
1195      DICON_ASSERT_PARSER_ID_( node.value.id(), literal_id );
1196      DICON_ASSERT_SIZE_( node.children.size(), 1 );
1197      DICON_ASSERT_PARSER_ID_( node.children.front().value.id(), literal_id );
1198
```

```
1199      enum
1200      { NORMAL
1201      , IN_SINGLE_QUOTE
1202      , IN_DOUBLE_QUOTE
1203      } state = NORMAL;
1204
1205      std::string literal;
1206
1207      BOOST_FOREACH( char x, std::string(node.children.front().value.
          begin(),
1208                                         node.children.front().value.end
          ()) )
1209      {
1210        switch( state ) {
1211        case NORMAL:
1212          switch( x ) {
1213          case '\'': state = IN_SINGLE_QUOTE; break;
1214          case '"': state = IN_DOUBLE_QUOTE; break;
1215          default: literal.push_back( x );
1216          }
1217          break;
1218
1219        case IN_SINGLE_QUOTE:
1220          switch( x ) {
1221          case '\'': state = NORMAL; break;
1222          default: literal.push_back( x );
1223          }
1224          break;
1225
1226        case IN_DOUBLE_QUOTE:
1227          switch( x ) {
1228          case '"': state = NORMAL; break;
1229          default: literal.push_back( x );
1230          }
1231          break;
1232        }
1233      }
1234
1235      if( state != NORMAL )
1236        BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<int>(
          state) );
1237
1238      return literal;
1239    }
1240
1241
1242    static
1243    rational_t
1244    parse_real( const tree_node<node_val_data<> > &node ) {
1245      DICON_ASSERT_PARSER_ID_( node.value.id(), real_id );
1246      DICON_ASSERT_SIZE_( node.children.size(), 1 );
1247      DICON_ASSERT_PARSER_ID_( node.children.front().value.id(), real_id
          );
1248
1249      return string_to_rational( std::string(node.children.front().value
          .begin(),
1250                                             node.children.front().value
          .end()) );
1251    }
1252
1253
```

```
1254    static
1255    LazySet<std::string>::ptr_t
1256    parse_argument( const tree_node<node_val_data<> > &node ) {
1257      DICON_ASSERT_PARSER_ID_( node.value.id(), argument_id );
1258      DICON_ASSERT_SIZE_( node.children.size(), 1 );
1259      DICON_ASSERT_PARSER_ID_( node.children.front().value.id(), set_id
          );
1260
1261      return parse_set( node.children.front() );
1262    }
1263
1264
1265    struct RationalToSimcount {
1266      typedef simcount_t result_type;
1267      typedef rational_t argument_type;
1268
1269      result_type
1270      operator()( argument_type x ) const {
1271        // Round x to nearest int.
1272        x += rational_t(sgn(x)) / 2;
1273        x = x.get_num() / x.get_den();
1274
1275        mpz_class min_value = boost::lexical_cast<mpz_class>( std::
          numeric_limits<simcount_t>::min() );
1276        mpz_class max_value = boost::lexical_cast<mpz_class>( std::
          numeric_limits<simcount_t>::max() );
1277
1278        assert( x.get_den() == 1 );
1279        if( min_value <= x.get_num()
1280            && x.get_num() <= max_value )
1281        {
1282          return boost::lexical_cast<result_type>( x.get_num() );
1283        }
1284        else {
1285          BOOST_THROW_EXCEPTION( SpecifierInvalidSimcountError()
1286                                 << errinfo_value<mpz_class>(x.get_num()
                                   )
1287                                 << errinfo_min_value<mpz_class>(
                                   min_value)
1288                                 << errinfo_max_value<mpz_class>(
                                   max_value) );
1289        }
1290      }
1291    };
1292
1293
1294    static
1295    LazySet<simcount_t>::ptr_t
1296    parse_schedule( const tree_node<node_val_data<> > &node ) {
1297      DICON_ASSERT_PARSER_ID_( node.value.id(), schedule_id );
1298      DICON_ASSERT_SIZE_MIN_( node.children.size(), 1 );
1299
1300      {
1301        const tree_node<node_val_data<> > &child = node.children.front()
          ;
1302
1303        if( child.value.id() == SpecifierGrammar::exp_range_id ) {
1304          return lazy_transform( lazy_transform(parse_exp_range(child),
1305                                                RealToRational())
1306                                 , RationalToSimcount()
1307                                 );
```

```
1308        }
1309        else if( child.value.id() == SpecifierGrammar::range_id ) {
1310          return lazy_transform( parse_range(child, true)
1311                                 , RationalToSimcount()
1312                                 );
1313        }
1314      }
1315
1316      std::set<rational_t> values;
1317
1318      BOOST_FOREACH( const tree_node<node_val_data<> > &child, node.children ) {
1319        if( child.value.id() == SpecifierGrammar::exp_or_real_id ) {
1320          const ExpressionOrReal &exp_or_real = parse_exp_or_real( child );
1321
1322          if( exp_or_real.is_real() )
1323            values.insert( exp_or_real.real() );
1324          else {
1325            if( !exp_or_real.expression().references().empty() ) {
1326              BOOST_THROW_EXCEPTION( SpecifierUndefinedVariableError()
1327                                     << errinfo_specifier_variable_name(*exp_or_real.expression().references().begin()) );
1328            }
1329
1330            values.insert( RealToRational()(exp_or_real.expression()()(LazyValue<real_t>::variables_t())) );
1331          }
1332        }
1333        else
1334          BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<parser_id>(child.value.id()) );
1335      }
1336
1337      return lazy_transform( lazy_container<std::vector<rational_t> >(values.begin(),
1338                                                                      values.end())
1339                             , RationalToSimcount()
1340                             );
1341    }
1342
1343
1344    template< typename T, int which >
1345    static
1346    typename LazySet<T>::ptr_t
1347    parse_specifier( const std::string &specifier, typename LazySet<T>::ptr_t (*parse)(const tree_node<node_val_data<> > &) ) {
1348      using namespace boost::spirit::classic;
1349      using namespace ::detail;
1350
1351      const tree_parse_info<> &info = pt_parse( specifier.c_str(), SpecifierGrammar().use_parser<which>() );
1352
1353      if( !info.full ) {
1354        BOOST_THROW_EXCEPTION( SpecifierParseError()
1355                               << errinfo_value<std::string>(specifier.substr(info.match ? info.length : 0)) );
1356      }
1357
1358      DICON_ASSERT_SIZE_( info.trees.size(), 1 );
1359
1360      return parse( info.trees.front() );
1361    }
1362
1363  }
1364
1365
1366  #undef DICON_ASSERT_PARSER_ID_
1367  #undef DICON_ASSERT_SIZE_
1368  #undef DICON_ASSERT_SIZE_MIN_
1369
1370
1371  LazySet<std::string>::ptr_t
1372  parse_argument_specifier( const std::string &specifier ) {
1373    return detail::parse_specifier<std::string, detail::SpecifierGrammar::argument>( specifier, detail::parse_argument );
1374  }
1375
1376
1377  LazySet<simcount_t>::ptr_t
1378  parse_schedule_specifier( const std::string &specifier ) {
1379    return detail::parse_specifier<simcount_t, detail::SpecifierGrammar::schedule>( specifier, detail::parse_schedule );
1380  }
```

Listing B.53: src/stats.hpp

```
1  #ifndef DICON_STATS_HPP_
2  #define DICON_STATS_HPP_
3
4  /*--- [Distribution] ---------------------------------------------------------
5   * This file is part of the Disease Control System DiCon.
6   *
7   * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8   * Designed and developed with the guidance of Nedialko B. Dimitrov
9   * and Lauren Ancel Meyers at the University of Texas at Austin.
10  *
11  * DiCon is free software: you can redistribute it and/or modify it
12  * under the terms of the GNU General Public License as published by
13  * the Free Software Foundation, either version 3 of the License, or
14  * (at your option) any later version.
15  *
16  * DiCon is distributed in the hope that it will be useful, but
17  * WITHOUT ANY WARRANTY; without even the implied warranty of
18  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19  * General Public License for more details.
20  *
21  * You should have received a copy of the GNU General Public License
22  * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23  *-----------------------------------------------------------------------------*/
24
25 /**
26  * @file
27  * @brief Statistics class.
28  */
29 #include <boost/noncopyable.hpp>
30 #include <boost/optional.hpp>
31 #include <map>
32 #include <ostream>
```

```
#include <stdint.h>
#include <sys/time.h>
#include <vector>


class StatsEntry;

/**
 * @brief Collect time statistics.
 *
 * The Statistics class collects time statistics showing how much time
 * the program spent in a certain state or function. This works in a
 * hierarchical way: at each point that an object of the Statistics
 * class exists, a scope can be defined by instantiating an object of
 * the StatsEntry class, referring to the Statistics object. The
 * Statistics object then keeps track of the wall clock time the
 * program ran while the StatsEntry object was still alive.
 *
 * This can be demonstrated as follows.
 *
 * @code
static Statistics statistics;

void function_a() {
  StatsEntry entry( statistics, "function_a()" );
  // Do some work here, taking some time.
}

void function_b() {
  StatsEntry entry( statistics, "function_b()" );
  // Do some work here, taking some time.
  // Call function_a() also.
}

int main() {
  StatsEntry entry( statistics, "main()" );
  // Do some work here, taking some time.
  // Call function_a() and function_b().
  statistics.print( std::cout );
}
@endcode
 *
 * This might result in an output like the following.
 *
 * @verbatim
name                        ticks     %/rel     %/tot            total
  time
(total)                         0     100.0     100.0       0 days
  00:00:35
  main()                        1     100.0     100.0       0 days
  00:00:35
    function_a()                1      28.6      28.6       0 days
  00:00:10
    function_b()                1      42.9      42.9       0 days
  00:00:15
      function_a()              1      66.7      28.6       0 days
  00:00:10
@endverbatim
 *
 * As can be seen in this example, the sum of all children need not be
 * equal to the total time spent in any given node in the statistics
 * tree. For this reason, an additional parameter can be given when
 * creating an object of the StatsEntry class, creating an additional
 * virtual node that appears to be active whenever no other child node
 * exists. If this were used in the example above, the output might
 * have looked like the following.
 *
 * @verbatim
name                        ticks     %/rel     %/tot            total
  time
(total)                         0     100.0     100.0       0 days
  00:00:35
  main()                        1     100.0     100.0       0 days
  00:00:35
    (rest)                      0      28.6      28.6       0 days
  00:00:10
    function_a()                1      28.6      28.6       0 days
  00:00:10
      (rest)                    0     100.0      28.6       0 days
  00:00:10
    function_b()                1      42.9      42.9       0 days
  00:00:15
      (rest)                    0      33.3      14.3       0 days
  00:00:05
      function_a()              1      66.7      28.6       0 days
  00:00:10
        (rest)                  0     100.0      28.6       0 days
  00:00:10
@endverbatim
 */
class Statistics
  : boost::noncopyable
{
  friend class StatsEntry;

public:
  /**
   * @brief Initialize statistics.
   *
   * Constructor that initializes a new statistics counter. This
   * starts the internal clock of the new Statistics object as
   * described in the description of the Statistics class.
   *
   * @throws TimeError when getting current time failed.
   */
  Statistics();

public:
  /**
   * @brief Output statistics.
   *
   * Output the collected time statistics to the given output
   * stream. The format might look similar to the one shown in the
   * description of the Statistics class, with lines for each node in
   * the statistics tree.
   *
   * The statistics object's internal state is kept frozen as long as
   * the output has not fully completed. This ensures that no
   * incoherent state is created. For this reason, this method might
   * throw an exception of type TimeError.
   *
   * @param output Output stream.
```

```
139     * @throws TimeError when freezing or resuming statistics failed.
140     */
141    void print( std::ostream &output );
142
143  protected:
144    /**
145     * @brief Push node onto tree.
146     *
147     * Push another node onto the statistics tree, at the current
148     * position given by previous calls to push() and pop(), or at the
149     * root node if no such call has been made so far. If the node (with
150     * name given by @e what) does not exist it will be created,
151     * otherwise the node's existing data will be updated.
152     *
153     * The priority (given by @e priority) can be used to define an
154     * order in which this node will be arranged relative to the other
155     * child nodes of its parent node, when printing the statistics with
156     * print(). If more than one child has the same priority, they will
157     * be arranged in the lexicographical order of the names given by @e
158     * what.
159     *
160     * If parameter @e idle is non-empty, an idle node is created
161     * whenever the new node has no active children. This is
162     * demonstrated by the examples given in the description of the
163     * Statistics class. This idle node can have its own priority given
164     * by @e idle_priority.
165     *
166     * @param what Node name.
167     * @param priority Node priority.
168     * @param idle Idle node name.
169     * @param idle_priority Idle node priority.
170     *
171     * @throws TimeError when getting current time failed.
172     */
173    void push( const std::string &what, int priority = 0,
174               const std::string &idle = std::string(), int
    ↪idle_priority = 0 );
175    /**
176     * @brief Pop node from tree.
177     *
178     * Pop the current node from the statistics tree. If the parent node
179     * had an idle node defined, it will automatically be pushed onto
180     * the tree in place of the current node.
181     *
182     * @throws TimeError when getting current time failed.
183     */
184    void pop();
185
186  private:
187    void push_( const std::pair<int, std::string> &entry, bool tick );
188    void pop_();
189
190  private:
191    class TimeRecord {
192    public:
193      TimeRecord();
194    public:
195      void start( bool tick = false );
196      void stop();
197    public:
198      double time() const;
199      uint64_t ticks() const;
200    private:
201      bool running_;
202      uint64_t ticks_;
203      double total_time_;
204      timeval last_start_;
205    };
206
207  private:
208    typedef std::vector<std::pair<int, std::string> > path_t;
209    typedef std::map<path_t, TimeRecord> records_t;
210
211    records_t records_;
212    path_t current_;
213    path_t idlers_;
214  };
215
216
217  /**
218   * @brief %Statistics helper.
219   *
220   * The StatsEntry class pushes a new node onto the statistics tree
221   * defined by the Statistics object given on construction, and
222   * automatically pops this node on destruction, as defined in the
223   * documentation of the Statistics class.
224   */
225  class StatsEntry
226    : boost::noncopyable
227  {
228  public:
229    /**
230     * @brief Push node onto tree.
231     *
232     * Constructor that pushes another node onto the statistics tree
233     * defined by the Statistics object given as @e stats, as defined in
234     * the documentation of the Statistics::push() method.
235     *
236     * @param stats Statistics object.
237     * @param what Node name.
238     * @param priority Node priority.
239     * @param idle Idle node name.
240     * @param idle_priority Idle node priority.
241     *
242     * @throws TimeError when getting current time failed.
243     */
244    StatsEntry( Statistics &stats, const std::string &what, int priority
    ↪ = 0,
245                const std::string &idle = std::string(), int
    ↪idle_priority = 0 );
246    /**
247     * @brief Pop node from tree.
248     *
249     * Destructor that pops the current node from the statistics tree
250     * defined by the Statistics object given on construction, as
251     * defined in the documentation of the Statistics::pop() method.
252     */
253    ~StatsEntry();
254
255  public:
256    /**
257     * @brief Stop entry.
```

```
258     *
259     * Stop the current statistics entry. This pops the current node
260     * from the statistics tree defined by the Statistics object given
261     * on construction, as defined in the documentation of the
262     * Statistics::pop() method, while also preventing the destructor of
263     * the StatsEntry from doing so. Thus, calling this method has the
264     * same effect as actually destroying the object. This can be useful
265     * in situations like the one in the following example.
266     *
267     * @code
268  StatsEntry entry_a( "entry_a()" );
269  // Do some time consuming task.
270  entry_a.stop();
271
272  StatsEntry entry_b( "entry_b()" );
273  // Do some time consuming task.
274  entry_b.stop();
275  @endcode
276     *
277     * Using the stop() method in this case allowed every StatsEntry
278     * object to remain in the same scope, freeing the programmer from
279     * the need of actually creating a new scope for each StatsEntry
280     * object used.
281     *
282     * @throws TimeError when getting current time failed.
283     */
284   void stop();
285
286 private:
287   bool stopped_;
288   Statistics &stats_;
289 };
290
291 #endif //DICON_STATS_HPP_
```

Listing B.54: src/stats.cpp

```
1  /*---- [Distribution] ------------------------------------------------
2   * This file is part of the Disease Control System DiCon.
3   *
4   * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
5   * Designed and developed with the guidance of Nedialko B. Dimitrov
6   * and Lauren Ancel Meyers at the University of Texas at Austin.
7   *
8   * DiCon is free software: you can redistribute it and/or modify it
9   * under the terms of the GNU General Public License as published by
10  * the Free Software Foundation, either version 3 of the License, or
11  * (at your option) any later version.
12  *
13  * DiCon is distributed in the hope that it will be useful, but
14  * WITHOUT ANY WARRANTY; without even the implied warranty of
15  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
16  * General Public License for more details.
17  *
18  * You should have received a copy of the GNU General Public License
19  * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
20  *--------------------------------------------------------------------*/
21
22 #include "stats.hpp"
23 #include "error.hpp"
24 #include <boost/foreach.hpp>
25 #include <boost/format.hpp>
26 #include <cerrno>
27 #include <cmath>
28
29
30 namespace detail {
31
32   static
33   timeval
34   time() {
35     struct timeval tv;
36
37     if( gettimeofday(&tv, NULL) != 0 )
38       BOOST_THROW_EXCEPTION( TimeError() << errinfo_api_function("gettimeofday") << errinfo_errno(errno) );
39
40     return tv;
41   }
42
43
44   static
45   double
46   time_difference( const timeval &a, const timeval &b ) {
47     return double(a.tv_sec) - double(b.tv_sec + 1) +
48       double(a.tv_usec + 1000000 - b.tv_usec) / 1000000;
49   }
50
51
52   static
53   std::string
54   time_to_str( double time ) {
55     double days = floor(time/60/60/24);
56     time -= days * 60*60*24;
57     double hours = floor(time/60/60);
58     time -= hours * 60*60;
59     double minutes = floor(time/60);
60     time -= minutes * 60;
61     double seconds = round(time);
62     time -= seconds;
63
64     return (boost::format("%3.0f_days_%02.0f:%02.0f:%02.0f") % days % hours % minutes % seconds).str();
65   }
66
67 }
68
69
70
71 Statistics::TimeRecord::TimeRecord()
72   : running_(false), ticks_(0), total_time_(0)
73 {
74 }
75
76
77 void
78 Statistics::TimeRecord::start( bool tick ) {
79   if( running_ )
80     BOOST_THROW_EXCEPTION( AssertionError() );
81
```

```
82    last_start_ = detail::time();
83    if( tick ) ++ticks_;
84    running_ = true;
85  }
86
87
88  void
89  Statistics::TimeRecord::stop() {
90    if( !running_ )
91      BOOST_THROW_EXCEPTION( AssertionError() );
92
93    total_time_ += std::max<double>( 0, detail::time_difference(detail::
    time(), last_start_) );
94    running_ = false;
95  }
96
97
98  double
99  Statistics::TimeRecord::time() const {
100   if( running_ )
101     BOOST_THROW_EXCEPTION( AssertionError() );
102
103   return total_time_;
104 }
105
106
107 uint64_t
108 Statistics::TimeRecord::ticks() const {
109   if( running_ )
110     BOOST_THROW_EXCEPTION( AssertionError() );
111
112   return ticks_;
113 }
114
115
116 Statistics::Statistics() {
117   records_[current_].start();
118 }
119
120
121 void
122 Statistics::print( std::ostream &output ) {
123   static const std::string root = "(total)";
124   static const size_t indent_per_level = 3;
125
126   path_t current = current_;
127
128   // Stop all records.
129
130   while( true ) {
131     records_[current].stop();
132     if( current.empty() )
133       break;
134     current.pop_back();
135   }
136
137   // Print out stats.
138
139   assert( current.empty() );
140   double total = records_.at(current).time();
141
142   size_t max_length = 0;
143   BOOST_FOREACH( const records_t::value_type &entry, records_ ) {
144     const path_t &path = entry.first;
145
146     max_length = std::max<size_t>( max_length,
147                                    path.size() * indent_per_level +
148                                    (path.empty() ? root : path.back().
    second).length() );
149   }
150
151   output << ( boost::format("name%s  ticks    %%/rel    %%/tot  
    total_time\n")
152              % std::string(max_length - std::string("name").length(), '
    ')
153              % std::string(detail::time_to_str(0).length() - std::
    string("total_time").length(), ' ')
154              );
155
156   BOOST_FOREACH( const records_t::value_type &entry, records_ ) {
157     const path_t &path = entry.first;
158     const TimeRecord &record = entry.second;
159
160     size_t indent = path.size() * indent_per_level;
161     const std::string &node = path.empty() ? root : path.back().second
    ;
162     double parent = path.empty() ? total
163                     : records_.at(path_t(path.begin(), --path.end())).
    time();
164
165     output << ( boost::format("%s%s%s    %9u    %5.1f    %5.1f    %s\n
    ")
166                 % std::string(indent, ' ')
167                 % node
168                 % std::string(max_length - node.length() - indent, ' ')
169                 % record.ticks()
170                 % (record.time() / parent * 100)
171                 % (record.time() / total * 100)
172                 % detail::time_to_str(record.time())
173                 );
174   }
175
176   // Restart records.
177
178   assert( current.empty() );
179   records_[current].start();
180   BOOST_FOREACH( const path_t::value_type &entry, current_ ) {
181     current.push_back( entry );
182     records_[current].start();
183   }
184 }
185
186
187 void
188 Statistics::push_( const std::pair<int, std::string> &entry, bool tick
    ) {
189   current_.push_back( entry );
190   records_[current_].start( tick );
191 }
192
193
194 void
```

```
195 Statistics::pop_() {
196   if( current_.empty() )
197     BOOST_THROW_EXCEPTION( AssertionError() );
198
199   records_[current_].stop();
200   current_.pop_back();
201 }
202
203
204 void
205 Statistics::push( const std::string &what, int priority, const std::
    string &idle, int idle_priority ) {
206   if( !idlers_.empty() && !idlers_.back().second.empty() )
207     pop_();
208
209   push_( std::make_pair(priority, what), true );
210
211   idlers_.push_back( std::make_pair(idle_priority, idle) );
212
213   if( !idlers_.back().second.empty() )
214     push_( idlers_.back(), false );
215 }
216
217
218 void
219 Statistics::pop() {
220   if( idlers_.empty() )
221     BOOST_THROW_EXCEPTION( AssertionError() );
222
223   if( !idlers_.back().second.empty() )
224     pop_();
225
226   pop_();
227
228   idlers_.pop_back();
229
230   if( !idlers_.empty() && !idlers_.back().second.empty() )
231     push_( idlers_.back(), false );
232 }
233
234
235 StatsEntry::StatsEntry( Statistics &stats,
236                         const std::string &what, int priority, const
    std::string &idle, int idle_priority )
237   : stopped_(false), stats_(stats)
238 {
239   stats_.push( what, priority, idle, idle_priority );
240 }
241
242
243 StatsEntry::~StatsEntry() {
244   try {
245     if( !stopped_ )
246       stats_.pop();
247   }
248   catch( ... ) {
249     // Ignore.
250   }
251 }
252
253
```

```
254 void
255 StatsEntry::stop() {
256   if( stopped_ )
257     BOOST_THROW_EXCEPTION( AssertionError() );
258
259   stats_.pop();
260
261   stopped_ = true;
262 }
```

Listing B.55: src/types.hpp

```
1  #ifndef DICON_TYPES_HPP_
2  #define DICON_TYPES_HPP_
3
4  /*--- [Distribution] ------------------------------------------------------
5   * This file is part of the Disease Control System DiCon.
6   *
7   * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8   * Designed and developed with the guidance of Nedialko B. Dimitrov
9   * and Lauren Ancel Meyers at the University of Texas at Austin.
10  *
11  * DiCon is free software: you can redistribute it and/or modify it
12  * under the terms of the GNU General Public License as published by
13  * the Free Software Foundation, either version 3 of the License, or
14  * (at your option) any later version.
15  *
16  * DiCon is distributed in the hope that it will be useful, but
17  * WITHOUT ANY WARRANTY; without even the implied warranty of
18  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19  * General Public License for more details.
20  *
21  * You should have received a copy of the GNU General Public License
22  * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23  *----------------------------------------------------------------------*/
24
25 /**
26  * @file
27  * @brief General types.
28  */
29 #include <stdint.h>
30 #include <string>
31 #include <vector>
32
33
34 /**
35  * @brief Log level.
36  *
37  * The @link ::LogLevel LogLevel@endlink enum defines the different
38  * log levels available in the main log system, as described in the
39  * log.hpp file.
40  */
41 enum LogLevel
42 { LOG_DEBUG   /// Debug messages.
43 , LOG_INFO    /// Info messages.
44 , LOG_WARNING /// Warning messages.
45 , LOG_ERROR   /// %Error messages.
46 , LOG_FAILURE /// Failure messages.
47 };
```

```
48
49
50 /// Convert textual representation to log level.
51 std::istream &operator>>( std::istream &in, LogLevel &result );
52
53
54 /**
55  * @brief %Job ID.
56  *
57  * The @link ::job_id_t job_id_t@endlink type defines the type used
58  * for job IDs. A valid job ID is a positive integer within the range
59  * defined by this type. The special value 0 can be used to indicate
60  * an error condition or an invalid job as it is not used for valid
61  * jobs.
62  *
63  * @see Job structure.
64  */
65 typedef size_t job_id_t;
66 /**
67  * @brief Simulation count.
68  *
69  * The @link ::simcount_t simcount_t@endlink type defines the type
70  * used for simulation counts. A valid simulation count is a
71  * non-negative integer within the range defined by this type. As
72  * there can easily exist more than 4 billion simulations within one
73  * optimization run (including system restarts after checkpointing),
74  * this type provides at least 64 bits.
75  */
76 typedef uint64_t simcount_t;
77
78 /// Argument set.
79 typedef std::vector<std::string> arguments_t;
80 /// Policy path.
81 typedef std::vector<std::string> policy_t;
82
83 #endif //DICON_TYPES_HPP_
```

Listing B.56: src/types.cpp

```
1  /*--- [Distribution] ---------------------------------------------------
2   * This file is part of the Disease Control System DiCon.
3   *
4   * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
5   * Designed and developed with the guidance of Nedialko B. Dimitrov
6   * and Lauren Ancel Meyers at the University of Texas at Austin.
7   *
8   * DiCon is free software: you can redistribute it and/or modify it
9   * under the terms of the GNU General Public License as published by
10  * the Free Software Foundation, either version 3 of the License, or
11  * (at your option) any later version.
12  *
13  * DiCon is distributed in the hope that it will be useful, but
14  * WITHOUT ANY WARRANTY; without even the implied warranty of
15  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
16  * General Public License for more details.
17  *
18  * You should have received a copy of the GNU General Public License
19  * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
20  *-----------------------------------------------------------------*/
21
22 #include "types.hpp"
23 #include <istream>
24
25
26 std::istream &
27 operator>>( std::istream &in, LogLevel &result ) {
28   std::string str;
29   in >> str;
30
31 #define DICON_LOG_LEVEL_STR_(STR, VAL) if( str == #STR ) do { result =
   LOG_##VAL; } while( false )
32   /**/ DICON_LOG_LEVEL_STR_( debug  , DEBUG   );
33   else DICON_LOG_LEVEL_STR_( info   , INFO    );
34   else DICON_LOG_LEVEL_STR_( warning, WARNING );
35   else DICON_LOG_LEVEL_STR_( error  , ERROR   );
36   else DICON_LOG_LEVEL_STR_( failure, FAILURE );
37   else in.setstate( std::ios_base::failbit );
38 #undef DICON_LOG_LEVEL_STR_
39
40   return in;
41 }
```

Listing B.57: src/util.hpp

```
1  #ifndef DICON_UTIL_HPP_
2  #define DICON_UTIL_HPP_
3
4  /*--- [Distribution] ---------------------------------------------------
5   * This file is part of the Disease Control System DiCon.
6   *
7   * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
8   * Designed and developed with the guidance of Nedialko B. Dimitrov
9   * and Lauren Ancel Meyers at the University of Texas at Austin.
10  *
11  * DiCon is free software: you can redistribute it and/or modify it
12  * under the terms of the GNU General Public License as published by
13  * the Free Software Foundation, either version 3 of the License, or
14  * (at your option) any later version.
15  *
16  * DiCon is distributed in the hope that it will be useful, but
17  * WITHOUT ANY WARRANTY; without even the implied warranty of
18  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
19  * General Public License for more details.
20  *
21  * You should have received a copy of the GNU General Public License
22  * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
23  *-----------------------------------------------------------------*/
24
25 /**
26  * @file
27  * @brief Utility functions.
28  */
29 #include "error.hpp"
30 #include <string>
31
32
33 /// Quoted-printable related error.
34 struct QPFormatError : virtual Error
```

```
35  { virtual const char *what() const throw() { return "Quoted-printable_
    related_error."; } };
36
37
38  /**
39   * @brief Decode quoted-printable string.
40   *
41   * Decode the string given by @e str in a variation of the
42   * quoted-printable encoding, as defined in the description of the
43   * qp_encode() function. If the data given by @e str is not of the
44   * expected format, this call will fail. This can happen when an
45   * equal sign (@c =) in the input is not followed by two hexadecimal
46   * digits (@c 0 to @c 9 or @c A to @c F or @c a to @c f).
47   *
48   * @param str Data to decode.
49   * @returns Original representation of data.
50   * @throws QPFormatError when @e str is invalid.
51   */
52  std::string qp_decode( const std::string &str );
53  /**
54   * @brief Encode quoted-printable string.
55   *
56   * Encode the string given by @e str in a variation of the
57   * quoted-printable encoding. In contrast to the original
58   * quoted-printable encoding as defined in RFC 2045, this function
59   * implements no special handling of line breaks, rather, it is to be
60   * used when the exact binary representation of the data given by @e
61   * str is to be encoded in a textual way containing only printable
62   * characters (ASCII range 32 to 126). The space character (ASCII 32)
63   * is also escaped whenever it appears at the beginning or at the end
64   * of the input string. Escaping is done as described in RFC 2045,
65   * using <code>=XX</code> sequences, where each @c X is a hexadecimal
66   * digit (@c 0 to @c 9 or @c A to @c F).
67   *
68   * If escaping of additional characters is needed, those can be given
69   * in the @e add parameter. Any character that appears in this string
70   * will also be escaped and not be used in the result. The only
71   * characters not allowed in this string are the hexadecimal digits
72   * (@c 0 to @c 9 and @c A to @c F) and the equal sign (@c =), as those
73   * are used for the quoted-printable encoding. If one of these
74   * characters is given, the behavior of this function is undefined.
75   *
76   * The following code shows some examples of the encoding.
77   *
78   * @code
79  assert( qp_encode("Test")        == "Test"        );
80  assert( qp_encode("A\nB")        == "A=0AB"       );
81  assert( qp_encode(" blank ")     == "=20blank=20" );
82  assert( qp_encode("A,B&C", ",&") == "A=2CB=26C"   );
83  @endcode
84   *
85   * @param str Data to encode.
86   * @param add Additional characters to escape.
87   * @returns Quoted-printable representation of data.
88   */
89  std::string qp_encode( const std::string &str, const std::string &add
    = std::string() );
90
91  #endif //DICON_UTIL_HPP_
```


Listing B.58: src/util.cpp

```
1   /*---- [Distribution] ------------------------------------------------
2    * This file is part of the Disease Control System DiCon.
3    *
4    * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
5    * Designed and developed with the guidance of Nedialko B. Dimitrov
6    * and Lauren Ancel Meyers at the University of Texas at Austin.
7    *
8    * DiCon is free software: you can redistribute it and/or modify it
9    * under the terms of the GNU General Public License as published by
10   * the Free Software Foundation, either version 3 of the License, or
11   * (at your option) any later version.
12   *
13   * DiCon is distributed in the hope that it will be useful, but
14   * WITHOUT ANY WARRANTY; without even the implied warranty of
15   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
16   * General Public License for more details.
17   *
18   * You should have received a copy of the GNU General Public License
19   * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
20   *--------------------------------------------------------------------*/
21
22  #include "util.hpp"
23
24  #include <boost/format.hpp>
25  #include <boost/scoped_array.hpp>
26  #include <cstring>
27  #include <stdexcept>
28
29
30  namespace detail {
31
32    static
33    char
34    hex_to_chr( unsigned char x, bool upcase = false ) {
35      DICON_ASSERT_RANGE<unsigned char>( x, 0, 15 );
36
37      if( 0 <= x && x < 10 )
38        return char( x + static_cast<unsigned char>('0') );
39      else
40        return char( (x-10) + static_cast<unsigned char>(upcase ? 'A' :
    'a') );
41    }
42
43
44    static
45    unsigned char
46    chr_to_hex( char x ) {
47      if( '0' <= x && x <= '9' )
48        return static_cast<unsigned char>(x) - static_cast<unsigned char
    >('0');
49      else if( 'a' <= x && x <= 'f' )
50        return static_cast<unsigned char>(x) - static_cast<unsigned char
    >('a') + 10;
51      else if( 'A' <= x && x <= 'F' )
52        return static_cast<unsigned char>(x) - static_cast<unsigned char
    >('A') + 10;
53      else
54        BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<char>(x
    ) );
```

```
  }

}



std::string
qp_decode( const std::string &str ) {
  std::string result;

  for( size_t i = 0; i < str.length(); ++i ) {
    unsigned char x = static_cast<unsigned char>( str[i] );

    if( x == 61 ) {
      if( i < str.length()-2 ) {
        try {
          x = detail::chr_to_hex(str[i+1]) * 16 + detail::chr_to_hex(str[i+2]);
          i += 2;
        }
        catch( boost::exception &e ) {
          BOOST_THROW_EXCEPTION( QPFormatError() << errinfo_value<std::string>(str.substr(i, 3))
                                 << errinfo_nested_exception(boost::copy_exception(e)) );
        }
      }
      else
        BOOST_THROW_EXCEPTION( QPFormatError() );
    }

    result += char( x );
  }

  return result;
}


std::string
qp_encode( const std::string &str, const std::string &add ) {
  std::string result;

  for( size_t i = 0; i < str.length(); ++i ) {
    unsigned char x = static_cast<unsigned char>( str[i] );

    if( (33 <= x && x <= 126 && x != 61 && add.find(x) == std::string::npos)
        || (x == 32 && i != 0 && i != (str.length() - 1)) )
    {
      // All printable ASCII characters (except '=' and those in add)
      // can be represented by themselves. Also spaces not at begin
      // or end of input string are represented by themselves.
      result += char( x );
    }
    else {
      // Escape according to quoted-printable encoding.
      result += '=';
      result += detail::hex_to_chr( x/16, true );
      result += detail::hex_to_chr( x%16, true );
    }
  }

  return result;
}
```

Listing B.59: src/version.hpp

```
#ifndef DICON_VERSION_HPP_
#define DICON_VERSION_HPP_

/*---- [Distribution] ------------------------------------------------------
 * This file is part of the Disease Control System DiCon.
 *
 * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
 * Designed and developed with the guidance of Nedialko B. Dimitrov
 * and Lauren Ancel Meyers at the University of Texas at Austin.
 *
 * DiCon is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * DiCon is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
 *--------------------------------------------------------------------------*/

/**
 * @file
 * @brief @link ::program_version program_version@endlink variable.
 */
/**
 * @brief Program version.
 *
 * The global @link ::program_version program_version@endlink variable
 * stores a textual representation of the current program version.
 * Version identifiers follow the versioning scheme
 * <em>year</em><code>.</code><em>month</em><code>.</code><em>release</em>,
 * where @e year is the 4-digit year and @e month is the 2-digit month
 * of release, and @e release is a non-negative incrementing integer
 * for situations where more than one version of the system is
 * released in the same month.  An exemplary program version is
 * <code>2009.12.0</code>.
 */
extern const char program_version[];

#endif //DICON_VERSION_HPP_
```

Listing B.60: src/version.cpp

```
/*---- [Distribution] ------------------------------------------------------
 * This file is part of the Disease Control System DiCon.
```

```
 3   *
 4   * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
 5   * Designed and developed with the guidance of Nedialko B. Dimitrov
 6   * and Lauren Ancel Meyers at the University of Texas at Austin.
 7   *
 8   * DiCon is free software: you can redistribute it and/or modify it
 9   * under the terms of the GNU General Public License as published by
10   * the Free Software Foundation, either version 3 of the License, or
11   * (at your option) any later version.
12   *
13   * DiCon is distributed in the hope that it will be useful, but
14   * WITHOUT ANY WARRANTY; without even the implied warranty of
15   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
16   * General Public License for more details.
17   *
18   * You should have received a copy of the GNU General Public License
19   * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
20   *-----------------------------------------------------------------*/
21
22  #include "version.hpp"
23
24  const char program_version[] = "2009.12.0";
```

Listing B.61: src/lazy/set.hpp

```
 1  #ifndef DICON_LAZY_SET_HPP_
 2  #define DICON_LAZY_SET_HPP_
 3
 4  /*---- [Distribution] ------------------------------------------------
 5   * This file is part of the Disease Control System DiCon.
 6   *
 7   * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
 8   * Designed and developed with the guidance of Nedialko B. Dimitrov
 9   * and Lauren Ancel Meyers at the University of Texas at Austin.
10   *
11   * DiCon is free software: you can redistribute it and/or modify it
12   * under the terms of the GNU General Public License as published by
13   * the Free Software Foundation, either version 3 of the License, or
14   * (at your option) any later version.
15   *
16   * DiCon is distributed in the hope that it will be useful, but
17   * WITHOUT ANY WARRANTY; without even the implied warranty of
18   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19   * General Public License for more details.
20   *
21   * You should have received a copy of the GNU General Public License
22   * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23   *-----------------------------------------------------------------*/
24
25  /**
26   * @file
27   * @brief LazySet interface.
28   */
29  #include <memory>
30
31
32  /**
33   * @brief Lazy set.
34   *
35   * The templated LazySet class provides the interface to an abstract
36   * lazy set. Such a set creates its elements when requested and has no
37   * predetermined length, i.e., it might be infinite. Methods are
38   * provided to check if another element in the set exists (has()), to
39   * create and return that element (get()), and to step to the next
40   * element in the set (inc()).
41   *
42   * The lazy set interface requires lazy sets to be immutable, i.e.,
43   * when reset to its original position (reset()) the lazy set has to
44   * return the same elements again. The set can be reset at any time;
45   * in fact, it has to be reset explicitly after construction as
46   * implementations of this interface are not required to start the set
47   * at its first element. Lazy sets must return their elements always
48   * in the same order.
49   *
50   * Due to the nature of an abstract base class, lazy sets are
51   * typically referenced to by pointer. The pointer class used and
52   * defined by @link LazySet::ptr_t ptr_t@endlink is the std::auto_ptr
53   * pointer. This allows for the efficient use of lazy sets, e.g., as
54   * constructor arguments as used in several of the implementations
55   * (such as LazyCombine, LazyCompose, LazyConcat, LazyTransform, and
56   * LazyUnion) while at the same time avoiding memory leaks.
57   *
58   * The following code shows how all elements in a lazy set can be
59   * printed.
60   *
61   * @code
62  LazySet<some_type>::ptr_t set = some_lazy_set;
63
64  for( set->reset(); set->has(); set->inc() ) {
65    std::cout << set->get() << std::endl;
66  }
67  @endcode
68   *
69   * For each lazy set implementation (LazyCombine, LazyCompose,
70   * LazyConcat, LazyContainer, LazyEmpty, LazyRange, LazySingleton,
71   * LazyTransform, or LazyUnion), there is a corresponding templated
72   * global helper function (lazy_combine(), lazy_compose(),
73   * lazy_concat(), lazy_container(), lazy_empty(), lazy_range(),
74   * lazy_singleton(), lazy_transform(), or lazy_union()) to create a
75   * new lazy set of that type and return a smart pointer to it. In
76   * most cases, these helper function can be automatically instantiated
77   * by the compiler as to avoid the necessity of explicitly specifying
78   * the template arguments. The following code demonstrates this.
79   *
80   * @code
81  LazySet<int>::ptr_t set = lazy_combine( lazy_range(10, 90, 10)
82                                        , lazy_range(9, 1, -1)
83                                        , std::plus<int>()
84                                        );
85  @endcode
86   *
87   * Notice how neither lazy_combine() nor lazy_range() required a
88   * template argument: the sets' value type was automatically deduced
89   * from the functions' arguments.  Also, the helper functions allowed
90   * for the exception-safe immediate use of the resulting sets as
91   * arguments to other lazy set constructors and helper functions.
92   *
93   * @note Although this class is named lazy @e set, it is rather a lazy
94   * @e list. In particular, a lazy set is allowed to contain the same
95   * element more than once.
```

```
 */
template< typename T >
class LazySet {
public:
  /// Set value type.
  typedef T value_t;
  /// Pointer to lazy set.
  typedef std::auto_ptr<LazySet> ptr_t;

public:
  virtual ~LazySet() {}

public:
  /**
   * @brief Clone lazy set.
   *
   * Clone the lazy set. The new set object must represent the exact
   * same set with the same elements as the original set. Also, it
   * must be independent from the original set, so that calling
   * reset() or inc() on either one set does not influence the other.
   *
   * @returns Copy of this set.
   */
  virtual ptr_t clone() const = 0;

public:
  /**
   * @brief Reset lazy set.
   *
   * Reset the lazy set to its first element. This method must be
   * called before first calling has(), get(), or inc(), as
   * implementations of the LazySet interface are not required to
   * start at the first element after construction.
   */
  virtual void reset() = 0;

public:
  /**
   * @brief Check if set has element.
   *
   * Check if the lazy set has another element. This function return
   * @c false if and only if the lazy set has reached its end. At this
   * point, calling get() or inc() is undefined.
   *
   * @returns @c true iff set has another element.
   */
  virtual bool has() const = 0;
  /**
   * @brief Get current element in set.
   *
   * Get the current element in the set. This creates and returns the
   * element at the current position within the set. The behavior of
   * calling this function when has() has returned @c false is not
   * defined.
   *
   * @returns Current element in set.
   */
  virtual T get() const = 0;
  /**
   * @brief Skip current element in set.
   *
   * Skip the current element in the set and go to the next. This
   * function increments the lazy set's internal iterator to the next
   * element in the set, skipping the current element. The behavior of
   * calling this function when has() has retured @c false is not
   * defined.
   */
  virtual void inc() = 0;
};


namespace detail {

  template< typename T >
  struct LazySetPtr {
    typedef typename T::element_type::value_t value_t;
    typedef typename LazySet<value_t>::ptr_t ptr_t;
  };

}

#endif //DICON_LAZY_SET_HPP_
```

Listing B.62: src/lazy/value.hpp

```
#ifndef DICON_LAZY_VALUE_HPP_
#define DICON_LAZY_VALUE_HPP_

/*--- [Distribution] ---------------------------------------------------
 * This file is part of the Disease Control System DiCon.
 *
 * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
 * Designed and developed with the guidance of Nedialko B. Dimitrov
 * and Lauren Ancel Meyers at the University of Texas at Austin.
 *
 * DiCon is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * DiCon is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
 *----------------------------------------------------------------------*/

/**
 * @file
 * @brief LazyValue interface.
 */
#include <map>
#include <memory>
#include <set>
#include <string>


/**
```

```
36   * @brief Lazy value.
37   *
38   * The templated LazyValue class provides the interface to an abstract
39   * lazy value. Such a value can be evaluated at any time providing a
40   * specific variable assignment. It also provides means to get the
41   * list of variables that are actually used by the lazy value.
42   *
43   * The lazy value interface requires the lazy value to be immutable,
44   * i.e., when evaluating it with the same variable assignment again at
45   * a later time, it has to evaluate to the same value.
46   *
47   * Due to the nature of an abstract base class, lazy values are
48   * typically referenced to by pointer. The pointer class used and
49   * defined by @link LazyValue::ptr_t ptr_t@endlink is the
50   * std::auto_ptr pointer. This allows for the efficient use of lazy
51   * values, e.g., as constructor arguments as used in several of the
52   * implementations (such as LazyBinaryOp, and LazyUnaryOp) while at
53   * the same time avoiding memory leaks.
54   *
55   * Variable names are always treated case-sensitive, i.e., an actual
56   * comparison for equality is performed by the @c string class's
57   * <code>operator=()</code> method. There is no restriction on
58   * variable names, i.e., even the empty string is a valid variable
59   * name.
60   *
61   * For each lazy value implementation (LazyBinaryOp, LazyConstant,
62   * LazyUnaryOp, or LazyVariable), there is a corresponding templated
63   * global helper function (lazy_binary_op(), lazy_constant(),
64   * lazy_unary_op(), or lazy_variable()) to create a new lazy value of
65   * that type and return a smart pointer to it. In most cases, these
66   * helper functions can be automatically instantiated by the compiler
67   * as to avoid the necessity of explicitly specifying template
68   * arguments. The following code demonstrates this.
69   *
70   * @code
71  LazyValue<int>::ptr_t value = lazy_binary_op( lazy_constant(42)
72                                              , lazy_binary_op( 
    lazy_variable<int>("x")
73                                                              , 
    lazy_variable<int>("y")
74                                                              , std::
    plus<int>()
75                                                              )
76                                              , std::minus<int>()
77                                              );
78  @endcode
79   *
80   * Notice how neither lazy_binary_op() nor lazy_constant() required a
81   * template argument: the lazy values' type was automatically deduced
82   * from the functions' arguments. Also, the helper functions allowed
83   * for the exception-safe immediate use of the resulting lazy values
84   * as arguments to other lazy value constructors and helper functions.
85   */
86  template< typename T >
87  class LazyValue {
88  public:
89    /// Value type.
90    typedef T value_t;
91    /// Pointer to lazy value.
92    typedef std::auto_ptr<LazyValue> ptr_t;
93    /// Set of variable names.
94    typedef std::set<std::string> references_t;
95    /// Variable assignment.
96    typedef std::map<std::string, T> variables_t;
97
98  public:
99    virtual ~LazyValue() {}
100
101 public:
102   /**
103    * @brief Get list of references.
104    *
105    * Get the list of variables actually referenced by the lazy
106    * value. This method must always return the same set since the lazy
107    * value interface requires lazy values to be immutable. See the
108    * description of the LazyValue class for details.
109    *
110    * @returns List of references.
111    */
112   virtual references_t references() const = 0;
113
114 public:
115   /**
116    * @brief Evaluate lazy value.
117    *
118    * Evaluate the lazy value, using the variable assignment given by
119    * @e variables. This method must always return the same value when
120    * called with the same variable assignment since the lazy value
121    * interface requires lazy values to be immutable. See the
122    * description of the LazyValue class for details.
123    *
124    * @param variables Variables assignment.
125    * @returns Evaluated lazy value.
126    */
127   virtual T operator()( const variables_t &variables ) const = 0;
128 };
129
130
131 namespace detail {
132
133   template< typename T >
134   struct LazyValuePtr {
135     typedef typename T::element_type::value_t value_t;
136     typedef typename LazyValue<value_t>::ptr_t ptr_t;
137   };
138
139 }
140
141 #endif //DICON_LAZY_VALUE_HPP_
```

Listing B.63: src/lazy/set/combine.hpp

```
1  #ifndef DICON_LAZY_SET_COMBINE_HPP_
2  #define DICON_LAZY_SET_COMBINE_HPP_
3
4  /*--- [Distribution] ---------------------------------------------------
5   * This file is part of the Disease Control System DiCon.
6   *
7   * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
8   * Designed and developed with the guidance of Nedialko B. Dimitrov
```

```
 9   * and Lauren Ancel Meyers at the University of Texas at Austin.
10   *
11   * DiCon is free software: you can redistribute it and/or modify it
12   * under the terms of the GNU General Public License as published by
13   * the Free Software Foundation, either version 3 of the License, or
14   * (at your option) any later version.
15   *
16   * DiCon is distributed in the hope that it will be useful, but
17   * WITHOUT ANY WARRANTY; without even the implied warranty of
18   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
19   * General Public License for more details.
20   *
21   * You should have received a copy of the GNU General Public License
22   * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
23   *-------------------------------------------------------------------*/
24
25  /**
26   * @file
27   * @brief LazyCombine class.
28   */
29  #include "../set.hpp"
30  #include <boost/noncopyable.hpp>
31  #include <functional>
32
33
34  /**
35   * @brief Combine lazy sets.
36   *
37   * The LazyCombine class implements the combination, or the Cartesian
38   * product, of two lazy sets. Thus, it represents the lazy set
39   * resulting from the application of a user-defined functor to all
40   * pairs of elements from the two original sets.
41   *
42   * The functor of type @e Op must be compatible with the following
43   * function type signature: <code>T(const S1 &, const S2 &)</code>.
44   *
45   * @tparam Op Functor to apply.
46   * @tparam T Value type of this set.
47   * @tparam S1 Value type of first original set.
48   * @tparam S2 Value type of second original set.
49   */
50  template< typename Op
51          , typename T = typename Op::result_type
52          , typename S1 = typename Op::first_argument_type
53          , typename S2 = typename Op::second_argument_type
54          >
55  class LazyCombine
56    : boost::noncopyable, public LazySet<T>
57  {
58  public:
59    /**
60     * @brief Create lazy combination set.
61     *
62     * Constructor that creates the lazy set that is the combination, or
63     * the Cartesian product, of the two lazy sets given by @e a and @e
64     * b. The resulting set contains the results from the application
65     * of the functor given by @e op to the pairs of values from sets @e
66     * a and @e b.
67     *
68     * The order of values in the resulting set is lexicographically
69     * given by the order of elements in sets @e a and @e b, e.g., if @e
70     * a had elements @c a1 and @c a2, and @e b had elements @c b1 and
71     * @c b2, the resulting set would contain the elements given by @e
72     * op(@c a1, @c b1), @e op(@c a1, @c b2), @e op(@c a2, @c b1), @e
73     * op(@c a2, @c b2), in that order.
74     *
75     * In case both @e a and @e b are finite sets, the resulting set
76     * contains exactly @e size(@e a) times @e size(@e b) elements,
77     * where @e size denotes the number of elements in the corresponding
78     * set. If either of the orignal sets is infinite, the resulting
79     * set will be infinite as well.
80     *
81     * In order to fulfill the lazy set interface, the functor @e op
82     * must return the same value each time it is called with the same
83     * arguments.
84     *
85     * @param a First set.
86     * @param b Second set.
87     * @param op Functor to apply.
88     */
89    LazyCombine( typename LazySet<S1>::ptr_t &a, typename LazySet<S2>::
    ptr_t &b, const Op &op = Op() );
90
91  public:
92    virtual typename LazySet<T>::ptr_t clone() const;
93
94  public:
95    virtual void reset();
96
97  public:
98    virtual bool has() const;
99    virtual T get() const;
100   virtual void inc();
101
102 private:
103   const typename LazySet<S1>::ptr_t a_;
104   const typename LazySet<S2>::ptr_t b_;
105   const Op op_;
106 };
107
108
109 /**
110  * @brief Create lazy combination set.
111  *
112  * Create lazy combination set as described in the documentation of
113  * the LazyCombine class. This templated helper function returns a
114  * smart pointer to the newly created set.
115  *
116  * @param a First set.
117  * @param b Second set.
118  * @param op Functor to apply.
119  * @returns Pointer to new lazy set.
120  */
121 template< typename Op, typename S1, typename S2 >
122 typename LazySet<typename Op::result_type>::ptr_t lazy_combine( S1 a,
    S2 b, const Op &op );
123
124
125 #include "combine.ipp"
126
127 #endif //DICON_LAZY_SET_COMBINE_HPP_
```

Listing B.64: src/lazy/set/combine.ipp

```
1  // -*- c++ -*-
2
3  /*------ [Distribution] ------------------------------------------------------
4   * This file is part of the Disease Control System DiCon.
5   *
6   * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
7   * Designed and developed with the guidance of Nedialko B. Dimitrov
8   * and Lauren Ancel Meyers at the University of Texas at Austin.
9   *
10  * DiCon is free software: you can redistribute it and/or modify it
11  * under the terms of the GNU General Public License as published by
12  * the Free Software Foundation, either version 3 of the License, or
13  * (at your option) any later version.
14  *
15  * DiCon is distributed in the hope that it will be useful, but
16  * WITHOUT ANY WARRANTY; without even the implied warranty of
17  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
18  * General Public License for more details.
19  *
20  * You should have received a copy of the GNU General Public License
21  * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
22  *----------------------------------------------------------------------------*/
23
24 template< typename Op, typename T, typename S1, typename S2 >
25 inline
26 LazyCombine<Op, T, S1, S2>::LazyCombine( typename LazySet<S1>::ptr_t &a, typename LazySet<S2>::ptr_t &b, const Op &op )
27   : a_(a), b_(b), op_(op)
28 {
29 }
30
31
32 template< typename Op, typename T, typename S1, typename S2 >
33 inline
34 typename LazySet<T>::ptr_t
35 LazyCombine<Op, T, S1, S2>::clone() const {
36   typename LazySet<S1>::ptr_t a = a_->clone();
37   typename LazySet<S2>::ptr_t b = b_->clone();
38
39   return typename LazySet<T>::ptr_t( new LazyCombine<Op, T, S1, S2>(a, b, op_) );
40 }
41
42
43 template< typename Op, typename T, typename S1, typename S2 >
44 inline
45 void
46 LazyCombine<Op, T, S1, S2>::reset() {
47   a_->reset();
48   b_->reset();
49 }
50
51
52 template< typename Op, typename T, typename S1, typename S2 >
53 inline
54 bool
55 LazyCombine<Op, T, S1, S2>::has() const {
56   return a_->has() && b_->has();
57 }
58
59
60 template< typename Op, typename T, typename S1, typename S2 >
61 inline
62 T
63 LazyCombine<Op, T, S1, S2>::get() const {
64   return op_( a_->get(), b_->get() );
65 }
66
67
68 template< typename Op, typename T, typename S1, typename S2 >
69 inline
70 void
71 LazyCombine<Op, T, S1, S2>::inc() {
72   b_->inc();
73
74   if( !b_->has() ) {
75     b_->reset();
76     a_->inc();
77   }
78 }
79
80
81 template< typename Op, typename S1, typename S2 >
82 inline
83 typename LazySet<typename Op::result_type>::ptr_t
84 lazy_combine( S1 a, S2 b, const Op &op ) {
85   return typename LazySet<typename Op::result_type>::ptr_t
86     ( new LazyCombine< Op
87                      , typename Op::result_type
88                      , typename detail::LazySetPtr<S1>::value_t
89                      , typename detail::LazySetPtr<S2>::value_t
90                      >(a, b, op)
91     );
92 }
```

Listing B.65: src/lazy/set/compose.hpp

```
1  #ifndef DICON_LAZY_SET_COMPOSE_HPP_
2  #define DICON_LAZY_SET_COMPOSE_HPP_
3
4  /*------ [Distribution] ------------------------------------------------------
5   * This file is part of the Disease Control System DiCon.
6   *
7   * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8   * Designed and developed with the guidance of Nedialko B. Dimitrov
9   * and Lauren Ancel Meyers at the University of Texas at Austin.
10  *
11  * DiCon is free software: you can redistribute it and/or modify it
12  * under the terms of the GNU General Public License as published by
13  * the Free Software Foundation, either version 3 of the License, or
14  * (at your option) any later version.
15  *
16  * DiCon is distributed in the hope that it will be useful, but
17  * WITHOUT ANY WARRANTY; without even the implied warranty of
18  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19  * General Public License for more details.
20  *
21  * You should have received a copy of the GNU General Public License
```

```
 * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
 *-------------------------------------------------------------------*/

/**
 * @file
 * @brief LazyCompose class.
 */
#include "../set.hpp"
#include <boost/noncopyable.hpp>
#include <functional>


/**
 * @brief Compose lazy sets.
 *
 * The LazyCompose class implements the composition of lazy sets. This
 * means that the given functor is called for each element in the
 * given lazy set, returning a lazy set each time. The composition
 * lazy set implemented by the LazyCompose class then is the
 * concatenation of these resulting sets.
 *
 * The functor of type @e Op must be compatible with the following
 * function type signature: <code>LazySet<T>::ptr_t(const S &)</code>.
 *
 * @tparam Op Functor to apply.
 * @tparam T Value type of this set.
 * @tparam S Value type of original set.
 */
template< typename Op
        , typename T = typename detail::LazySetPtr<typename Op::result_type>::value_t
        , typename S = typename Op::argument_type
        >
class LazyCompose
  : boost::noncopyable, public LazySet<T>
{
public:
  /**
   * @brief Create lazy composition set.
   *
   * Constructor that creates the lazy set that is the composition of
   * the functor given by @e op with the elements of the lazy set
   * given by @e x.
   *
   * The resulting lazy set will be infinite if and only if at least
   * one of the lazy sets returned by @e op, called for each element
   * in the original set @e x, is infinite.
   *
   * In order to fulfill the lazy set interface, the functor @e op
   * must return a lazy set containing the same value each time it is
   * called with the same argument.
   *
   * @param x Original lazy set.
   * @param op Functor to apply.
   */
  LazyCompose( typename LazySet<S>::ptr_t &x, const Op &op = Op() );

public:
  virtual typename LazySet<T>::ptr_t clone() const;

public:
  virtual void reset();

public:
  virtual bool has() const;
  virtual T get() const;
  virtual void inc();

private:
  void fetch() const;

private:
  const typename LazySet<S>::ptr_t x_;
  mutable typename LazySet<T>::ptr_t y_;
  const Op op_;
};


/**
 * @brief Create lazy composition set.
 *
 * Create lazy composition set as described in the documentation of
 * the LazyCompose class. This templated helper function returns a
 * smart pointer to the newly created set.
 *
 * @param x Original lazy set.
 * @param op Functor to apply.
 * @returns Pointer to new lazy set.
 */
template< typename Op, typename S >
typename detail::LazySetPtr<typename Op::result_type>::ptr_t lazy_compose( S x, const Op &op );


#include "compose.ipp"

#endif //DICON_LAZY_SET_COMPOSE_HPP_
```

Listing B.66: src/lazy/set/compose.ipp

```
// -*- c++ -*-

/*---- [Distribution] ------------------------------------------------*
 * This file is part of the Disease Control System DiCon.
 *
 * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
 * Designed and developed with the guidance of Nedialko B. Dimitrov
 * and Lauren Ancel Meyers at the University of Texas at Austin.
 *
 * DiCon is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * DiCon is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
```

```
 * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
 *-------------------------------------------------------------------*/

#include "../../error.hpp"


template< typename Op, typename T, typename S >
inline
LazyCompose<Op, T, S>::LazyCompose( typename LazySet<S>::ptr_t &x,
    const Op &op )
  : x_(x), op_(op)
{
}


template< typename Op, typename T, typename S >
inline
typename LazySet<T>::ptr_t
LazyCompose<Op, T, S>::clone() const {
  typename LazySet<S>::ptr_t x = x_->clone();

  return typename LazySet<T>::ptr_t( new LazyCompose<Op, T, S>(x, op_)
  );
}


template< typename Op, typename T, typename S >
inline
void
LazyCompose<Op, T, S>::reset() {
  x_->reset();

  fetch();
}


template< typename Op, typename T, typename S >
inline
bool
LazyCompose<Op, T, S>::has() const {
  return y_.get() && y_->has();
}


template< typename Op, typename T, typename S >
inline
T
LazyCompose<Op, T, S>::get() const {
  if( !y_.get() )
    BOOST_THROW_EXCEPTION( AssertionError() );

  return y_->get();
}


template< typename Op, typename T, typename S >
inline
void
LazyCompose<Op, T, S>::inc() {
  if( !y_.get() )
    BOOST_THROW_EXCEPTION( AssertionError() );

  y_->inc();

  if( !y_->has() )
    fetch();
}


template< typename Op, typename T, typename S >
inline
void
LazyCompose<Op, T, S>::fetch() const {
  do {
    if( x_->has() ) {
      y_ = op_( x_->get() );
      y_->reset();
      x_->inc();
    }
    else
      break;
  } while( !y_->has() );
}


template< typename Op, typename S >
inline
typename detail::LazySetPtr<typename Op::result_type>::ptr_t
lazy_compose( S x, const Op &op ) {
  return typename detail::LazySetPtr<typename Op::result_type>::ptr_t
    ( new LazyCompose< Op
                     , typename detail::LazySetPtr<typename Op::
result_type>::value_t
                     , typename detail::LazySetPtr<S>::value_t
                     >(x, op)
    );
}
```

Listing B.67: src/lazy/set/concat.hpp

```
#ifndef DICON_LAZY_SET_CONCAT_HPP_
#define DICON_LAZY_SET_CONCAT_HPP_

/*--- [Distribution] ------------------------------------------------*
 * This file is part of the Disease Control System DiCon.
 *
 * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
 * Designed and developed with the guidance of Nedialko B. Dimitrov
 * and Lauren Ancel Meyers at the University of Texas at Austin.
 *
 * DiCon is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * DiCon is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * General Public License for more details.
 *
```

```
 * You should have received a copy of the GNU General Public License
 * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
 *-----------------------------------------------------------------*/

/**
 * @file
 * @brief LazyConcat class.
 */
#include "../set.hpp"
#include <boost/noncopyable.hpp>


/**
 * @brief Concatenate lazy sets.
 *
 * The LazyConcat class implements the concatenation of two lazy sets.
 * Thus, it represents the lazy set that first contains all elements
 * of the first set, followed by all elements of the second set.
 *
 * @tparam T Value type of this and both original sets.
 */
template< typename T >
class LazyConcat
  : boost::noncopyable, public LazySet<T>
{
public:
  /**
   * @brief Create lazy concatenation set.
   *
   * Constructor that creates the lazy set that is the concatenation
   * of the two lazy sets given by @e a and @e b. The resulting set
   * first contains all elements of @e a, followed by all elements of
   * @e b.
   *
   * The resulting set will be infinite if and only if at least one of
   * the two sets @e a and @e b is infinite.
   *
   * @param a First set.
   * @param b Second set.
   */
  LazyConcat( typename LazySet<T>::ptr_t &a, typename LazySet<T>::ptr_t &b );

public:
  virtual typename LazySet<T>::ptr_t clone() const;

public:
  virtual void reset();

public:
  virtual bool has() const;
  virtual T get() const;
  virtual void inc();

private:
  const typename LazySet<T>::ptr_t a_;
  const typename LazySet<T>::ptr_t b_;
};


/**
 * @brief Create lazy concatenation set.
 *
 * Create lazy concatenation set as described in the documentation of
 * the LazyConcat class. This templated helper function returns a
 * smart pointer to the newly created set.
 *
 * @param a First set.
 * @param b Second set.
 * @returns Pointer to new lazy set.
 */
template< typename T >
typename detail::LazySetPtr<T>::ptr_t lazy_concat( T a, T b );


#include "concat.ipp"

#endif //DICON_LAZY_SET_CONCAT_HPP_
```

Listing B.68: src/lazy/set/concat.ipp

```
// -*- c++ -*-

/*---- [Distribution] ------------------------------------------------
 * This file is part of the Disease Control System DiCon.
 *
 * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
 * Designed and developed with the guidance of Nedialko B. Dimitrov
 * and Lauren Ancel Meyers at the University of Texas at Austin.
 *
 * DiCon is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * DiCon is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
 *-----------------------------------------------------------------*/

template< typename T >
inline
LazyConcat<T>::LazyConcat( typename LazySet<T>::ptr_t &a, typename LazySet<T>::ptr_t &b )
  : a_(a), b_(b)
{
}


template< typename T >
inline
typename LazySet<T>::ptr_t
LazyConcat<T>::clone() const {
  typename LazySet<T>::ptr_t a = a_->clone();
  typename LazySet<T>::ptr_t b = b_->clone();

```

```
39    return typename LazySet<T>::ptr_t( new LazyConcat<T>(a, b) );
40  }
41
42
43  template< typename T >
44  inline
45  void
46  LazyConcat<T>::reset() {
47    a_->reset();
48    b_->reset();
49  }
50
51
52  template< typename T >
53  inline
54  bool
55  LazyConcat<T>::has() const {
56    return a_->has() || b_->has();
57  }
58
59
60  template< typename T >
61  inline
62  T
63  LazyConcat<T>::get() const {
64    if( a_->has() )
65      return a_->get();
66    else
67      return b_->get();
68  }
69
70
71  template< typename T >
72  inline
73  void
74  LazyConcat<T>::inc() {
75    if( a_->has() )
76      a_->inc();
77    else
78      b_->inc();
79  }
80
81
82  template< typename T >
83  inline
84  typename detail::LazySetPtr<T>::ptr_t
85  lazy_concat( T a, T b ) {
86    return typename detail::LazySetPtr<T>::ptr_t
87      ( new LazyConcat<typename detail::LazySetPtr<T>::value_t>(a, b) );
88  }
```

Listing B.69: src/lazy/set/container.hpp

```
1   #ifndef DICON_LAZY_SET_CONTAINER_HPP_
2   #define DICON_LAZY_SET_CONTAINER_HPP_
3
4   /*--- [Distribution] ------------------------------------------------
5    * This file is part of the Disease Control System DiCon.
6    *
7    * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8    * Designed and developed with the guidance of Nedialko B. Dimitrov
9    * and Lauren Ancel Meyers at the University of Texas at Austin.
10   *
11   * DiCon is free software: you can redistribute it and/or modify it
12   * under the terms of the GNU General Public License as published by
13   * the Free Software Foundation, either version 3 of the License, or
14   * (at your option) any later version.
15   *
16   * DiCon is distributed in the hope that it will be useful, but
17   * WITHOUT ANY WARRANTY; without even the implied warranty of
18   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19   * General Public License for more details.
20   *
21   * You should have received a copy of the GNU General Public License
22   * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23   *-----------------------------------------------------------------*/
24
25  /**
26   * @file
27   * @brief LazyContainer class.
28   */
29  #include "../set.hpp"
30
31
32  /**
33   * @brief Lazy container set.
34   *
35   * The LazyContainer class implements the lazy set equivalent of the
36   * given container. Thus, it returns all the elements of the original
37   * container, in the order the container returns them. The lazy set
38   * stores the elements it contains in an internal container the type
39   * of which can be specified by the template argument @e C. The
40   * container class must fulfill either the sequence requirements or
41   * the associative container requirements specified by the standard.
42   *
43   * @tparam C Container type used.
44   * @tparam T Value type of this set.
45   */
46  template< typename C
47          , typename T = typename C::value_type
48          >
49  class LazyContainer
50    : public LazySet<T>
51  {
52  public:
53    /**
54     * @brief Create lazy set of container.
55     *
56     * Constructor that creates the lazy set that contains exactly the
57     * elements of the given container, in the same order as the
58     * container returns them. The internal container of the lazy set
59     * will be copy-constructed from the container given by @e
60     * container.
61     *
62     * @param container Original container.
63     */
64    LazyContainer( const C &container );
65
66    /**
67     * @brief Create lazy set of container.
```

```
68      *
69      * Constructor that creates the lazy set that contains exactly the
70      * elements of the given container range, in the order they are
71      * given in the range. The internal container of the lazy set will
72      * be constructed from this range, i.e., the original container from
73      * which this range was obtained can be destroyed after this
74      * constructor returns.
75      *
76      * @param begin Iterator to first element.
77      * @param end Iterator past last element.
78      */
79     template< typename InputIterator >
80     LazyContainer( InputIterator begin, InputIterator end );
81
82   public:
83     virtual typename LazySet<T>::ptr_t clone() const;
84
85   public:
86     virtual void reset();
87
88   public:
89     virtual bool has() const;
90     virtual T get() const;
91     virtual void inc();
92
93   private:
94     const C container_;
95     typename C::const_iterator position_;
96 };
97
98
99   /**
100   * @brief Create lazy container set.
101   *
102   * Create lazy container set as described in the documentation of the
103   * LazyContainer class. This templated helper function returns a smart
104   * pointer to the newly created set.
105   *
106   * @param container Original container.
107   * @returns Pointer to new lazy set.
108   */
109  template< typename C >
110  typename LazySet<typename C::value_type>::ptr_t lazy_container( const
     C &container );
111
112  /**
113   * @brief Create lazy container set.
114   *
115   * Create lazy container set as described in the documentation of the
116   * LazyContainer class. This templated helper function returns a smart
117   * pointer to the newly created set.
118   *
119   * @param begin Iterator to first element.
120   * @param end Iterator past last element.
121   * @returns Pointer to new lazy set.
122   */
123  template< typename C, typename InputIterator >
124  typename LazySet<typename C::value_type>::ptr_t lazy_container(
     InputIterator begin, InputIterator end );
125
126
```



```
127  #include "container.ipp"
128
129  #endif //DICON_LAZY_SET_CONTAINER_HPP_
```

Listing B.70: src/lazy/set/container.ipp

```
1   // -*- c++ -*-
2
3   /*---- [Distribution] ------------------------------------------------
4    * This file is part of the Disease Control System DiCon.
5    *
6    * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
7    * Designed and developed with the guidance of Nedialko B. Dimitrov
8    * and Lauren Ancel Meyers at the University of Texas at Austin.
9    *
10   * DiCon is free software: you can redistribute it and/or modify it
11   * under the terms of the GNU General Public License as published by
12   * the Free Software Foundation, either version 3 of the License, or
13   * (at your option) any later version.
14   *
15   * DiCon is distributed in the hope that it will be useful, but
16   * WITHOUT ANY WARRANTY; without even the implied warranty of
17   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
18   * General Public License for more details.
19   *
20   * You should have received a copy of the GNU General Public License
21   * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
22   *--------------------------------------------------------------------*/
23
24  #include "../../error.hpp"
25
26
27  template< typename C, typename T >
28  inline
29  LazyContainer<C, T>::LazyContainer( const C &container )
30    : container_(container), position_(container_.begin())
31  {
32  }
33
34
35  template< typename C, typename T >
36  template< typename InputIterator >
37  inline
38  LazyContainer<C, T>::LazyContainer( InputIterator begin, InputIterator
     end )
39    : container_(begin, end), position_(container_.begin())
40  {
41  }
42
43
44  template< typename C, typename T >
45  inline
46  typename LazySet<T>::ptr_t
47  LazyContainer<C, T>::clone() const {
48    return typename LazySet<T>::ptr_t( new LazyContainer<C, T>(
     container_) );
49  }
50
51
```

```
52 template< typename C, typename T >
53 inline
54 void
55 LazyContainer<C, T>::reset() {
56   position_ = container_.begin();
57 }
58
59
60 template< typename C, typename T >
61 inline
62 bool
63 LazyContainer<C, T>::has() const {
64   return position_ != container_.end();
65 }
66
67
68 template< typename C, typename T >
69 inline
70 T
71 LazyContainer<C, T>::get() const {
72   return *position_;
73 }
74
75
76 template< typename C, typename T >
77 inline
78 void
79 LazyContainer<C, T>::inc() {
80   ++position_;
81 }
82
83
84 template< typename C >
85 inline
86 typename LazySet<typename C::value_type>::ptr_t
87 lazy_container( const C &container ) {
88   return typename LazySet<typename C::value_type>::ptr_t
89     ( new LazyContainer<C, typename C::value_type>(container) );
90 }
91
92
93 template< typename C, typename InputIterator >
94 inline
95 typename LazySet<typename C::value_type>::ptr_t
96 lazy_container( InputIterator begin, InputIterator end ) {
97   return typename LazySet<typename C::value_type>::ptr_t
98     ( new LazyContainer<C, typename C::value_type>(begin, end) );
99 }
```

Listing B.71: src/lazy/set/empty.hpp

```
1  #ifndef DICON_LAZY_SET_EMPTY_HPP_
2  #define DICON_LAZY_SET_EMPTY_HPP_
3
4  /*--- [Distribution] ---------------------------------------------
5   * This file is part of the Disease Control System DiCon.
6   *
7   * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8   * Designed and developed with the guidance of Nedialko B. Dimitrov
9   * and Lauren Ancel Meyers at the University of Texas at Austin.
10  *
11  * DiCon is free software: you can redistribute it and/or modify it
12  * under the terms of the GNU General Public License as published by
13  * the Free Software Foundation, either version 3 of the License, or
14  * (at your option) any later version.
15  *
16  * DiCon is distributed in the hope that it will be useful, but
17  * WITHOUT ANY WARRANTY; without even the implied warranty of
18  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19  * General Public License for more details.
20  *
21  * You should have received a copy of the GNU General Public License
22  * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23  *-----------------------------------------------------------------*/
24
25 /**
26  * @file
27  * @brief LazyEmpty class.
28  */
29 #include "../set.hpp"
30
31
32 /**
33  * @brief Lazy empty set.
34  *
35  * The LazyEmpty class implements an empty lazy set. Thus, it does
36  * not contain any elements, and calling get() or inc() always
37  * fails. It can be useful as a starting point for building bigger
38  * lazy sets using LazyCombine, LazyConcat, or LazyUnion.
39  *
40  * @note Though this empty lazy set does not contain any elements, it
41  *   must still have a value type associated. This is required both in
42  *   order to satisfy the LazySet interface (which requires a value
43  *   type) and to make the helper functions described in the LazySet
44  *   interface easier to use.
45  *
46  * @tparam T Value type of this set.
47  */
48 template< typename T >
49 class LazyEmpty
50   : public LazySet<T>
51 {
52 public:
53   /**
54    * @brief Create lazy empty set.
55    *
56    * Constructor that creates the lazy set that does not contain any
57    * elements. The resulting set is empty, and calling get() or inc()
58    * always fails, while has() always returns @c false.
59    */
60   LazyEmpty();
61
62 public:
63   virtual typename LazySet<T>::ptr_t clone() const;
64
65 public:
66   virtual void reset();
67
68 public:
69   virtual bool has() const;
```

```
70    virtual T get() const;
71    virtual void inc();
72  };
73
74
75  /**
76   * @brief Create lazy empty set.
77   *
78   * Create lazy empty set as described in the documentation of the
79   * LazyEmpty class. This templated helper function returns a smart
80   * pointer to the newly created set.
81   *
82   * @returns Pointer to new lazy set.
83   */
84  template< typename T >
85  typename LazySet<T>::ptr_t lazy_empty();
86
87
88  #include "empty.ipp"
89
90  #endif //DICON_LAZY_SET_EMPTY_HPP_
```

Listing B.72: src/lazy/set/empty.ipp

```
1   // -*- c++ -*-
2
3   /*---- [Distribution] ------------------------------------------------
4    * This file is part of the Disease Control System DiCon.
5    *
6    * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
7    * Designed and developed with the guidance of Nedialko B. Dimitrov
8    * and Lauren Ancel Meyers at the University of Texas at Austin.
9    *
10   * DiCon is free software: you can redistribute it and/or modify it
11   * under the terms of the GNU General Public License as published by
12   * the Free Software Foundation, either version 3 of the License, or
13   * (at your option) any later version.
14   *
15   * DiCon is distributed in the hope that it will be useful, but
16   * WITHOUT ANY WARRANTY; without even the implied warranty of
17   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
18   * General Public License for more details.
19   *
20   * You should have received a copy of the GNU General Public License
21   * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
22   *--------------------------------------------------------------------*/
23
24  #include "../../error.hpp"
25
26
27  template< typename T >
28  inline
29  LazyEmpty<T>::LazyEmpty() {
30    // Nothing to do.
31  }
32
33
34  template< typename T >
35  inline
36  typename LazySet<T>::ptr_t
37  LazyEmpty<T>::clone() const {
38    return typename LazySet<T>::ptr_t( new LazyEmpty<T>() );
39  }
40
41
42  template< typename T >
43  inline
44  void
45  LazyEmpty<T>::reset() {
46    // Nothing to do.
47  }
48
49
50  template< typename T >
51  inline
52  bool
53  LazyEmpty<T>::has() const {
54    return false;
55  }
56
57
58  template< typename T >
59  inline
60  T
61  LazyEmpty<T>::get() const {
62    BOOST_THROW_EXCEPTION( AssertionError() );
63  }
64
65
66  template< typename T >
67  inline
68  void
69  LazyEmpty<T>::inc() {
70    BOOST_THROW_EXCEPTION( AssertionError() );
71  }
72
73
74  template< typename T >
75  inline
76  typename LazySet<T>::ptr_t
77  lazy_empty() {
78    return typename LazySet<T>::ptr_t
79      ( new LazyEmpty<T>() );
80  }
```

Listing B.73: src/lazy/set/range.hpp

```
1   #ifndef DICON_LAZY_SET_RANGE_HPP_
2   #define DICON_LAZY_SET_RANGE_HPP_
3
4   /*---- [Distribution] ------------------------------------------------
5    * This file is part of the Disease Control System DiCon.
6    *
7    * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
8    * Designed and developed with the guidance of Nedialko B. Dimitrov
9    * and Lauren Ancel Meyers at the University of Texas at Austin.
10   *
11   * DiCon is free software: you can redistribute it and/or modify it
```

```
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * DiCon is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
 *----------------------------------------------------------------------*/

/**
 * @file
 * @brief LazyRange class.
 */
#include "../set.hpp"


/**
 * @brief Lazy range set.
 *
 * The LazyRange class implements the infinite (unbounded) or finite
 * (bounded) range given by a first value and a step size, or a first
 * value, a step size, and a last value.
 *
 * The value type of the set should represent a numerical value (such
 * as @c int, @c double, or some custom class). It must implement both
 * comparison operators (@c =, @c <, and @c <=) that define a total
 * order on the value type, and the addition of values as well as the
 * multiplication with an integer of type @c size_t, i.e., for @c t
 * and @c s of type @e T and @c i of type @c size_t, the expressions
 * <code>t+s</code> and <code>t*i</code> must be defined and
 * convertible to type @e T.
 *
 * @note In order to avoid precision errors with floating point types,
 *   the actual calculation performed by the lazy set implementation
 *   when getting one of its elements is <code>lower+step*i</code>
 *   where @c i is of type @c size_t. This should be taken into
 *   account when considering performance.
 *
 * @tparam T Value type of this set.
 */
template< typename T >
class LazyRange
  : public LazySet<T>
{
public:
  /**
   * @brief Create unbounded lazy range set.
   *
   * Constructor that creates the lazy set that represents the
   * infinite (unbounded) range with elements @e lower, @e lower + @e
   * step, @e lower + @e step + @e step, ...
   *
   * @param lower First element in lazy set.
   * @param step Step between elements.
   * @throws AssertionError when @e step is zero (i.e.,
   *   <code>=T()</code>).
   */
  LazyRange( const T &lower, const T &step );
  /**
   * @brief Create bounded lazy range set.
   *
   * Constructor that creates the lazy set that represents the finite
   * (bounded) range with elements @e lower, @e lower + @e step, @e
   * lower + @e step + @e step, ..., @e upper. If @e step is positive
   * (i.e., <code>>T()</code>), the last element in the set is the one
   * that is still less than or equal to @e upper, and the set
   * contains no value greater than @e upper. If @e step is negative
   * (i.e., <code><T()</code>), the last element in the set is the one
   * that is still greater than or equal to @e upper, and the set
   * contains no value less than @e upper.
   *
   * @note Despite the names it is not an error when @e upper is less
   *   than @e lower. In fact this might be useful when @e step is a
   *   negative value.
   *
   * @param lower First element in lazy set.
   * @param upper Final element in lazy set.
   * @param step Step between elements.
   * @throws AssertionError when @e step is zero (i.e.,
   *   <code>=T()</code>).
   */
  LazyRange( const T &lower, const T &upper, const T &step );

public:
  virtual typename LazySet<T>::ptr_t clone() const;

public:
  virtual void reset();

public:
  virtual bool has() const;
  virtual T get() const;
  virtual void inc();

private:
  const bool open_;
  const T lower_;
  const T upper_;
  const T step_;
  size_t pos_;
};


/**
 * @brief Create lazy range set.
 *
 * Create lazy range set as described in the documentation of the
 * LazyRange class. This templated helper function returns a smart
 * pointer to the newly created set.
 *
 * @param lower First element in lazy set.
 * @param step Step between elements.
 * @returns Pointer to new lazy set.
 * @throws AssertionError when @e step is zero (i.e.,
 *   <code>=T()</code>).
 */
template< typename T >
```

```
133 typename LazySet<T>::ptr_t lazy_range( const T &lower, const T &step );
134
135 /**
136  * @brief Create lazy range set.
137  *
138  * Create lazy range set as described in the documentation of the
139  * LazyRange class. This templated helper function returns a smart
140  * pointer to the newly created set.
141  *
142  * @param lower First element in lazy set.
143  * @param upper Final element in lazy set.
144  * @param step Step between elements.
145  * @returns Pointer to new lazy set.
146  * @throws AssertionError when @e step is zero (i.e.,
147  *   <code>=T()</code>).
148  */
149 template< typename T >
150 typename LazySet<T>::ptr_t lazy_range( const T &lower, const T &upper,
    const T &step );
151
152
153 #include "range.ipp"
154
155 #endif //DICON_LAZY_SET_RANGE_HPP_
```

Listing B.74: src/lazy/set/range.ipp

```
1  // -*- c++ -*-
2
3  /*---- [Distribution] ------------------------------------------------
4   * This file is part of the Disease Control System DiCon.
5   *
6   * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
7   * Designed and developed with the guidance of Nedialko B. Dimitrov
8   * and Lauren Ancel Meyers at the University of Texas at Austin.
9   *
10  * DiCon is free software: you can redistribute it and/or modify it
11  * under the terms of the GNU General Public License as published by
12  * the Free Software Foundation, either version 3 of the License, or
13  * (at your option) any later version.
14  *
15  * DiCon is distributed in the hope that it will be useful, but
16  * WITHOUT ANY WARRANTY; without even the implied warranty of
17  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
18  * General Public License for more details.
19  *
20  * You should have received a copy of the GNU General Public License
21  * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
22  *--------------------------------------------------------------------*/
23
24 #include "../../error.hpp"
25
26
27 template< typename T >
28 inline
29 LazyRange<T>::LazyRange( const T &lower, const T &step )
30   : open_(true), lower_(lower), upper_(T()), step_(step)
31 {
32   if( !(step_ < T() || T() < step_) )
33     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<T>(step) );
34 }
35
36
37 template< typename T >
38 inline
39 LazyRange<T>::LazyRange( const T &lower, const T &upper, const T &step )
40   : open_(false), lower_(lower), upper_(upper), step_(step)
41 {
42   if( !(step_ < T() || T() < step_) )
43     BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<T>(step) );
44 }
45
46
47 template< typename T >
48 inline
49 typename LazySet<T>::ptr_t
50 LazyRange<T>::clone() const {
51   return typename LazySet<T>::ptr_t( open_
52                                      ? new LazyRange<T>(lower_, step_)
53                                      : new LazyRange<T>(lower_, upper_, step_) );
54 }
55
56
57 template< typename T >
58 inline
59 void
60 LazyRange<T>::reset() {
61   pos_ = 0;
62 }
63
64
65 template< typename T >
66 inline
67 bool
68 LazyRange<T>::has() const {
69   if( open_ )
70     return true;
71   else {
72     return T() < step_
73       ? get() <= upper_
74       : upper_ <= get();
75   }
76 }
77
78
79 template< typename T >
80 inline
81 T
82 LazyRange<T>::get() const {
83   return lower_ + T(pos_ * step_);
84 }
85
86
87 template< typename T >
88 inline
```

```
89  void
90  LazyRange<T>::inc() {
91    ++pos_;
92  }
93
94
95  template< typename T >
96  inline
97  typename LazySet<T>::ptr_t
98  lazy_range( const T &lower, const T &step ) {
99    return typename LazySet<T>::ptr_t
100     ( new LazyRange<T>(lower, step) );
101 }
102
103
104 template< typename T >
105 inline
106 typename LazySet<T>::ptr_t
107 lazy_range( const T &lower, const T &upper, const T &step ) {
108   return typename LazySet<T>::ptr_t
109     ( new LazyRange<T>(lower, upper, step) );
110 }
```

Listing B.75: src/lazy/set/singleton.hpp

```
1   #ifndef DICON_LAZY_SET_SINGLETON_HPP_
2   #define DICON_LAZY_SET_SINGLETON_HPP_
3
4   /*--- [Distribution] ---------------------------------------------------
5    * This file is part of the Disease Control System DiCon.
6    *
7    * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8    * Designed and developed with the guidance of Nedialko B. Dimitrov
9    * and Lauren Ancel Meyers at the University of Texas at Austin.
10   *
11   * DiCon is free software: you can redistribute it and/or modify it
12   * under the terms of the GNU General Public License as published by
13   * the Free Software Foundation, either version 3 of the License, or
14   * (at your option) any later version.
15   *
16   * DiCon is distributed in the hope that it will be useful, but
17   * WITHOUT ANY WARRANTY; without even the implied warranty of
18   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19   * General Public License for more details.
20   *
21   * You should have received a copy of the GNU General Public License
22   * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23   *-----------------------------------------------------------------*/
24
25  /**
26   * @file
27   * @brief LazySingleton class.
28   */
29  #include "../set.hpp"
30
31
32  /**
33   * @brief Lazy singleton set.
34   *
35   * The LazySingleton class implements the lazy set that contains
36   * exactly one element, given on construction.  The lazy set
37   * implementation keeps a copy of the value given on construction
38   * after the constructor returns.
39   *
40   * @tparam T Value type of this set.
41   */
42  template< typename T >
43  class LazySingleton
44    : public LazySet<T>
45  {
46  public:
47    /**
48     * @brief Create lazy singleton set.
49     *
50     * Constructor that creates the lazy set that contains only the
51     * single element given by @e value.
52     *
53     * @param value Only element in lazy set.
54     */
55    LazySingleton( const T &value );
56
57  public:
58    virtual typename LazySet<T>::ptr_t clone() const;
59
60  public:
61    virtual void reset();
62
63  public:
64    virtual bool has() const;
65    virtual T get() const;
66    virtual void inc();
67
68  private:
69    const T value_;
70    bool done_;
71  };
72
73
74  /**
75   * @brief Create lazy singleton set.
76   *
77   * Create lazy singleton set as described in the documentation of the
78   * LazySingleton class. This templated helper function returns a smart
79   * pointer to the newly created set.
80   *
81   * @param value Only element in lazy set.
82   * @returns Pointer to new lazy set.
83   */
84  template< typename T >
85  typename LazySet<T>::ptr_t lazy_singleton( const T &value );
86
87
88  #include "singleton.ipp"
89
90  #endif //DICON_LAZY_SET_SINGLETON_HPP_
```

Listing B.76: src/lazy/set/singleton.ipp

```
// -*- c++ -*-

/*---- [Distribution] ------------------------------------------------
 * This file is part of the Disease Control System DiCon.
 *
 * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
 * Designed and developed with the guidance of Nedialko B. Dimitrov
 * and Lauren Ancel Meyers at the University of Texas at Austin.
 *
 * DiCon is free software: you  can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * DiCon  is distributed in  the hope  that it  will be  useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
 *--------------------------------------------------------------------*/

#include "../../error.hpp"


template< typename T >
inline
LazySingleton<T>::LazySingleton( const T &value )
  : value_(value)
{
}


template< typename T >
inline
typename LazySet<T>::ptr_t
LazySingleton<T>::clone() const {
  return typename LazySet<T>::ptr_t( new LazySingleton<T>(value_) );
}


template< typename T >
inline
void
LazySingleton<T>::reset() {
  done_ = false;
}


template< typename T >
inline
bool
LazySingleton<T>::has() const {
  return !done_;
}


template< typename T >
inline
T
LazySingleton<T>::get() const {
  if( !done_ )
    return value_;
  else
    BOOST_THROW_EXCEPTION( AssertionError() );
}


template< typename T >
inline
void
LazySingleton<T>::inc() {
  if( !done_ )
    done_ = true;
  else
    BOOST_THROW_EXCEPTION( AssertionError() );
}


template< typename T >
inline
typename LazySet<T>::ptr_t
lazy_singleton( const T &value ) {
  return typename LazySet<T>::ptr_t
    ( new LazySingleton<T>(value) );
}
```

Listing B.77: src/lazy/set/transform.hpp

```
#ifndef DICON_LAZY_SET_TRANSFORM_HPP_
#define DICON_LAZY_SET_TRANSFORM_HPP_

/*---- [Distribution] ------------------------------------------------
 * This file is part of the Disease Control System DiCon.
 *
 * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
 * Designed and developed with the guidance of Nedialko B. Dimitrov
 * and Lauren Ancel Meyers at the University of Texas at Austin.
 *
 * DiCon is free software: you  can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * DiCon  is distributed in  the hope  that it  will be  useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
 *--------------------------------------------------------------------*/

/**
 * @file
 * @brief LazyTransform class.
 */
#include "../set.hpp"
#include <boost/noncopyable.hpp>
```

```
31 #include <functional>
32
33
34 /**
35  * @brief Transform lazy set.
36  *
37  * The LazyTransform class implements the transformation of the given
38  * lazy set. Thus, it contains exactly the values returned by the
39  * application of the given functor to all elements from the original
40  * set.
41  *
42  * The functor of type @e Op must be compatible with the following
43  * function type signature: <code>T(const S &)</code>.
44  *
45  * @tparam Op Functor to apply.
46  * @tparam T Value type of this set.
47  * @tparam S Value type of original set.
48  */
49 template< typename Op
50         , typename T = typename Op::result_type
51         , typename S = typename Op::argument_type
52         >
53 class LazyTransform
54   : boost::noncopyable, public LazySet<T>
55 {
56 public:
57   /**
58    * @brief Create lazy transformation set.
59    *
60    * Constructor that creates the lazy set that is the transformation
61    * of the lazy set given by @e x. The resulting set contains the
62    * results from the application of the functor given by @e op to the
63    * values from set @e x.
64    *
65    * The resulting set contains exactly as many elements as the
66    * original set given by @e x. Therefore, it is finite if and only
67    * if the original set @e x is finite.
68    *
69    * In order to fulfill the lazy set interface, the functor @e op
70    * must return the same value each time it is called with the same
71    * argument.
72    *
73    * @param x Original set.
74    * @param op Functor to apply.
75    */
76   LazyTransform( typename LazySet<S>::ptr_t &x, const Op &op = Op() );
77
78 public:
79   virtual typename LazySet<T>::ptr_t clone() const;
80
81 public:
82   virtual void reset();
83
84 public:
85   virtual bool has() const;
86   virtual T get() const;
87   virtual void inc();
88
89 private:
90   const typename LazySet<S>::ptr_t x_;
91   const Op op_;
92 };
93
94
95 /**
96  * @brief Create lazy transformation set.
97  *
98  * Create lazy transformation set as described in the documentation of
99  * the LazyTransform class. This templated helper function returns a
100 * smart pointer to the newly created set.
101 *
102 * @param x Original set.
103 * @param op Functor to apply.
104 * @returns Pointer to new lazy set.
105 */
106 template< typename Op, typename S >
107 typename LazySet<typename Op::result_type>::ptr_t lazy_transform( S x,
    const Op &op );
108
109
110 #include "transform.ipp"
111
112 #endif //DICON_LAZY_SET_TRANSFORM_HPP_
```

Listing B.78: src/lazy/set/transform.ipp

```
1  // -*- c++ -*-
2
3  /*---- [Distribution] ------------------------------------------------
4   * This file is part of the Disease Control System DiCon.
5   *
6   * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
7   * Designed and developed with the guidance of Nedialko B. Dimitrov
8   * and Lauren Ancel Meyers at the University of Texas at Austin.
9   *
10  * DiCon is free software: you can redistribute it and/or modify it
11  * under the terms of the GNU General Public License as published by
12  * the Free Software Foundation, either version 3 of the License, or
13  * (at your option) any later version.
14  *
15  * DiCon is distributed in the hope that it will be useful, but
16  * WITHOUT ANY WARRANTY; without even the implied warranty of
17  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
18  * General Public License for more details.
19  *
20  * You should have received a copy of the GNU General Public License
21  * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
22  *--------------------------------------------------------------------*/
23
24 template< typename Op, typename T, typename S >
25 inline
26 LazyTransform<Op, T, S>::LazyTransform( typename LazySet<S>::ptr_t &x,
    const Op &op )
27   : x_(x), op_(op)
28 {
29 }
30
31
32 template< typename Op, typename T, typename S >
33 inline
```

```
34 typename LazySet<T>::ptr_t
35 LazyTransform<Op, T, S>::clone() const {
36   typename LazySet<S>::ptr_t x = x_->clone();
37
38   return typename LazySet<T>::ptr_t( new LazyTransform<Op, T, S>(x,
     op_) );
39 }
40
41
42 template< typename Op, typename T, typename S >
43 inline
44 void
45 LazyTransform<Op, T, S>::reset() {
46   x_->reset();
47 }
48
49
50 template< typename Op, typename T, typename S >
51 inline
52 bool
53 LazyTransform<Op, T, S>::has() const {
54   return x_->has();
55 }
56
57
58 template< typename Op, typename T, typename S >
59 inline
60 T
61 LazyTransform<Op, T, S>::get() const {
62   return op_( x_->get() );
63 }
64
65
66 template< typename Op, typename T, typename S >
67 inline
68 void
69 LazyTransform<Op, T, S>::inc() {
70   x_->inc();
71 }
72
73
74 template< typename Op, typename S >
75 inline
76 typename LazySet<typename Op::result_type>::ptr_t
77 lazy_transform( S x, const Op &op ) {
78   return typename LazySet<typename Op::result_type>::ptr_t
79     ( new LazyTransform< Op
80                        , typename Op::result_type
81                        , typename detail::LazySetPtr<S>::value_t
82                        >(x, op)
83     );
84 }
```

Listing B.79: src/lazy/set/union.hpp

```
1 #ifndef DICON_LAZY_SET_UNION_HPP_
2 #define DICON_LAZY_SET_UNION_HPP_
3
4 /*--- [Distribution] ---------------------------------------------
5  * This file is part of the Disease Control System DiCon.
6  *
7  * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8  * Designed and developed with the guidance of Nedialko B. Dimitrov
9  * and Lauren Ancel Meyers at the University of Texas at Austin.
10 *
11 * DiCon is free software: you  can redistribute it and/or modify it
12 * under the terms of the GNU General Public License as published by
13 * the Free Software Foundation, either version 3 of the License, or
14 * (at your option) any later version.
15 *
16 * DiCon  is distributed in  the hope  that it  will be  useful, but
17 * WITHOUT ANY WARRANTY; without  even the  implied  warranty  of
18 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19 * General Public License for more details.
20 *
21 * You should have received a copy of the GNU General Public License
22 * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23 *-----------------------------------------------------------------*/
24
25 /**
26  * @file
27  * @brief LazyUnion class.
28  */
29 #include "../set.hpp"
30 #include <boost/noncopyable.hpp>
31
32
33 /**
34  * @brief Unite lazy sets.
35  *
36  * The LazyUnion class  implements the set union of  two lazy sets. It
37  * requires that  both of  the original sets  are sorted  according to
38  * some strict order defined by  the comparison operators (@c =, @c <,
39  * @c <=) on @e T. The elements in the resulting set are also sorted.
40  *
41  * The implementation works as follows.  If the current element in the
42  * first original set were @c a  and the current element in the second
43  * original set were  @c b, then the current  element in the resulting
44  * set  would  be  @c  a  if  <code>a<=b</code>  or  @c  b  if
45  * <code>a>b</code>. When taking a step with inc(), the first original
46  * set  would be  stepped to  its next  element  if <code>a<=b</code>,
47  * while the second original set  would be stepped to its next element
48  * if <code>a>=b</code>,  i.e., if <code>a=b</code>  the element would
49  * end up only once in the resulting set.
50  *
51  * Neither  the first  nor the  second  original set  is filtered  for
52  * duplicate elements.   If either set  contains an element  more than
53  * once,  it will also show up multiple times in the resulting set.
54  *
55  * @tparam T Value type of this set.
56  */
57 template< typename T >
58 class LazyUnion
59   : boost::noncopyable, public LazySet<T>
60 {
61 public:
62   /**
63    * @brief Create lazy union set.
64    *
65    * Constructor that  creates the lazy set  that is the  set union of
```

```
66     * the two lazy sets given by @e a and @e b. Both original sets must
67     * be sorted as described in the documentation of the LazyUnion
68     * class.
69     *
70     * @param a First set.
71     * @param b Second set.
72     */
73   LazyUnion( typename LazySet<T>::ptr_t &a, typename LazySet<T>::ptr_t
       &b );
74
75   public:
76     virtual typename LazySet<T>::ptr_t clone() const;
77
78   public:
79     virtual void reset();
80
81   public:
82     virtual bool has() const;
83     virtual T get() const;
84     virtual void inc();
85
86   private:
87     const typename LazySet<T>::ptr_t a_;
88     const typename LazySet<T>::ptr_t b_;
89 };
90
91
92 /**
93  * @brief Create lazy union set.
94  *
95  * Create lazy union set as described in the documentation of the
96  * LazyUnion class. This templated helper function returns a smart
97  * pointer to the newly created set.
98  *
99  * @param a First set.
100 * @param b Second set.
101 * @returns Pointer to new lazy set.
102 */
103 template< typename T >
104 typename detail::LazySetPtr<T>::ptr_t lazy_union( T a, T b );
105
106
107 #include "union.ipp"
108
109 #endif //DICON_LAZY_SET_UNION_HPP_
```

Listing B.80: src/lazy/set/union.ipp

```
1  // -*- c++ -*-
2
3  /*---- [Distribution] ------------------------------------------------
4   * This file is part of the Disease Control System DiCon.
5   *
6   * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
7   * Designed and developed with the guidance of Nedialko B. Dimitrov
8   * and Lauren Ancel Meyers at the University of Texas at Austin.
9   *
10  * DiCon is free software: you can redistribute it and/or modify it
11  * under the terms of the GNU General Public License as published by
12  * the Free Software Foundation, either version 3 of the License, or
13  * (at your option) any later version.
14  *
15  * DiCon is distributed in the hope that it will be useful, but
16  * WITHOUT ANY WARRANTY; without even the implied warranty of
17  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
18  * General Public License for more details.
19  *
20  * You should have received a copy of the GNU General Public License
21  * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
22  *--------------------------------------------------------------------*/
23
24 template< typename T >
25 inline
26 LazyUnion<T>::LazyUnion( typename LazySet<T>::ptr_t &a, typename
      LazySet<T>::ptr_t &b )
27   : a_(a), b_(b)
28 {
29 }
30
31
32 template< typename T >
33 inline
34 typename LazySet<T>::ptr_t
35 LazyUnion<T>::clone() const {
36   typename LazySet<T>::ptr_t a = a_->clone();
37   typename LazySet<T>::ptr_t b = b_->clone();
38
39   return typename LazySet<T>::ptr_t( new LazyUnion<T>(a, b) );
40 }
41
42
43 template< typename T >
44 inline
45 void
46 LazyUnion<T>::reset() {
47   a_->reset();
48   b_->reset();
49 }
50
51
52 template< typename T >
53 inline
54 bool
55 LazyUnion<T>::has() const {
56   return a_->has() || b_->has();
57 }
58
59
60 template< typename T >
61 inline
62 T
63 LazyUnion<T>::get() const {
64   if( a_->has() && b_->has() ) {
65     const T &a = a_->get();
66     const T &b = b_->get();
67
68     if( a < b )
69       return a;
70     else if( b < a )
71       return b;
```

```
72      else
73        return a;
74    }
75    else if( a_->has() )
76      return a_->get();
77    else
78      return b_->get();
79  }
80
81
82  template< typename T >
83  inline
84  void
85  LazyUnion<T>::inc() {
86    if( a_->has() && b_->has() ) {
87      const T &a = a_->get();
88      const T &b = b_->get();
89
90      if( a < b )
91        a_->inc();
92      else if( b < a )
93        b_->inc();
94      else {
95        a_->inc();
96        b_->inc();
97      }
98    }
99    else if( a_->has() )
100     a_->inc();
101   else
102     b_->inc();
103 }
104
105
106 template< typename T >
107 inline
108 typename detail::LazySetPtr<T>::ptr_t
109 lazy_union( T a, T b ) {
110   return typename detail::LazySetPtr<T>::ptr_t
111     ( new LazyUnion<typename detail::LazySetPtr<T>::value_t>(a, b) );
112 }
```

Listing B.81: src/lazy/value/binary.hpp

```
1   #ifndef DICON_LAZY_VALUE_BINARY_HPP_
2   #define DICON_LAZY_VALUE_BINARY_HPP_
3
4   /*--- [Distribution] ---------------------------------------------------
5    * This file is part of the Disease Control System DiCon.
6    *
7    * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8    * Designed and developed with the guidance of Nedialko B. Dimitrov
9    * and Lauren Ancel Meyers at the University of Texas at Austin.
10   *
11   * DiCon is free software: you can redistribute it and/or modify it
12   * under the terms of the GNU General Public License as published by
13   * the Free Software Foundation, either version 3 of the License, or
14   * (at your option) any later version.
15   *
16   * DiCon is distributed in the hope that it will be useful, but
17   * WITHOUT ANY WARRANTY; without even the implied warranty of
18   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19   * General Public License for more details.
20   *
21   * You should have received a copy of the GNU General Public License
22   * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23   *-----------------------------------------------------------------------*/
24
25  /**
26   * @file
27   * @brief LazyBinaryOp class.
28   */
29  #include "../value.hpp"
30  #include <boost/noncopyable.hpp>
31
32
33  /**
34   * @brief Lazy binary operator.
35   *
36   * The LazyBinaryOp implements the lazy application of a binary
37   * functor to two lazy values.
38   *
39   * The functor of type @e Op must be compatible with the following
40   * function type signature: <code>T(const T &, const T &)</code>.
41   *
42   * @tparam Op Functor to apply.
43   * @tparam T Lazy value's type.
44   */
45  template< typename Op, typename T = typename Op::result_type >
46  class LazyBinaryOp
47    : boost::noncopyable, public LazyValue<T>
48  {
49  public:
50    /**
51     * @brief Create lazy binary operator.
52     *
53     * Constructor that creates the lazy value that is the lazy
54     * application of the binary functor given by @e op to the two lazy
55     * values given by @e a and @e b. The resulting lazy value will
56     * depend on all variables the original lazy values depend on.
57     *
58     * In order to fulfill the lazy value interface, the functor @e op
59     * must return the same value each time it is called with the same
60     * arguments.
61     *
62     * @param a First lazy value.
63     * @param b Second lazy value.
64     * @param op Functor to apply lazily.
65     */
66    LazyBinaryOp( typename LazyValue<T>::ptr_t &a, typename LazyValue<T
    >::ptr_t &b, const Op &op = Op() );
67
68  public:
69    virtual typename LazyValue<T>::references_t references() const;
70
71  public:
72    virtual T operator()( const typename LazyValue<T>::variables_t &
    variables ) const;
73
74  private:
```

```
75    const typename LazyValue<T>::ptr_t a_;
76    const typename LazyValue<T>::ptr_t b_;
77    Op op_;
78  };
79
80
81  /**
82   * @brief Create lazy binary operator.
83   *
84   * Create lazy  binary operator as  described in the  documentation of
85   * the LazyBinaryOp  class. This  templated helper function  returns a
86   * smart pointer to the newly created lazy value.
87   *
88   * @param a First lazy value.
89   * @param b Second lazy value.
90   * @param op Functor to apply lazily.
91   * @returns Pointer to new lazy value.
92   */
93  template< typename Op, typename T >
94  typename detail::LazyValuePtr<T>::ptr_t lazy_binary_op( T a, T b,
     const Op &op );
95
96
97  #include "binary.ipp"
98
99  #endif //DICON_LAZY_VALUE_BINARY_HPP_
```

Listing B.82: src/lazy/value/binary.ipp

```
1   // -*- c++ -*-
2
3   /*---- [Distribution] ------------------------------------------------
4    * This file is part of the Disease Control System DiCon.
5    *
6    * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
7    * Designed and developed with the guidance of Nedialko B. Dimitrov
8    * and Lauren Ancel Meyers at the University of Texas at Austin.
9    *
10   * DiCon is free software: you  can redistribute it and/or modify it
11   * under the terms of the GNU General Public License as published by
12   * the Free Software Foundation, either version 3 of the License, or
13   * (at your option) any later version.
14   *
15   * DiCon  is distributed in  the hope  that it  will be  useful, but
16   * WITHOUT ANY WARRANTY;  without  even the  implied  warranty  of
17   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
18   * General Public License for more details.
19   *
20   * You should have received a copy of the GNU General Public License
21   * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
22   *--------------------------------------------------------------------*/
23
24  template< typename Op, typename T >
25  inline
26  LazyBinaryOp<Op, T>::LazyBinaryOp( typename LazyValue<T>::ptr_t &a,
     typename LazyValue<T>::ptr_t &b, const Op &op )
27    : a_(a), b_(b), op_(op)
28  {
29  }
30
31
32  template< typename Op, typename T >
33  inline
34  typename LazyValue<T>::references_t
35  LazyBinaryOp<Op, T>::references() const {
36    typename LazyValue<T>::references_t references;
37
38    const typename LazyValue<T>::references_t &ref_a = a_->references();
39    const typename LazyValue<T>::references_t &ref_b = b_->references();
40
41    references.insert( ref_a.begin(), ref_a.end() );
42    references.insert( ref_b.begin(), ref_b.end() );
43
44    return references;
45  }
46
47
48  template< typename Op, typename T >
49  inline
50  T
51  LazyBinaryOp<Op, T>::operator()( const typename LazyValue<T>::
     variables_t &variables ) const {
52    return op_( (*a_)(variables), (*b_)(variables) );
53  }
54
55
56  template< typename Op, typename T >
57  inline
58  typename detail::LazyValuePtr<T>::ptr_t
59  lazy_binary_op( T a, T b, const Op &op ) {
60    return typename detail::LazyValuePtr<T>::ptr_t
61      ( new LazyBinaryOp< Op
62                        , typename detail::LazyValuePtr<T>::value_t
63                        >(a, b, op)
64      );
65  }
```

Listing B.83: src/lazy/value/constant.hpp

```
1   #ifndef DICON_LAZY_VALUE_CONSTANT_HPP_
2   #define DICON_LAZY_VALUE_CONSTANT_HPP_
3
4   /*---- [Distribution] ------------------------------------------------
5    * This file is part of the Disease Control System DiCon.
6    *
7    * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8    * Designed and developed with the guidance of Nedialko B. Dimitrov
9    * and Lauren Ancel Meyers at the University of Texas at Austin.
10   *
11   * DiCon is free software: you  can redistribute it and/or modify it
12   * under the terms of the GNU General Public License as published by
13   * the Free Software Foundation, either version 3 of the License, or
14   * (at your option) any later version.
15   *
16   * DiCon  is distributed in  the hope  that it  will be  useful, but
17   * WITHOUT ANY WARRANTY;  without  even the  implied  warranty  of
18   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19   * General Public License for more details.
```

```
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
 *-----------------------------------------------------------------*/

/**
 * @file
 * @brief LazyConstant class.
 */
#include "../value.hpp"


/**
 * @brief Lazy constant value.
 *
 * The LazyConstant class implements a constant lazy value. Thus, it
 * does not depend an any variables and will always evaluate to the
 * same value.
 *
 * @tparam T Lazy value's type.
 */
template< typename T >
class LazyConstant
  : public LazyValue<T>
{
public:
  /**
   * @brief Create lazy constant value.
   *
   * Constructor that creates the lazy value that is constant and
   * always evaluates to the value given by @e value.
   *
   * @param value Constant lazy value.
   */
  LazyConstant( const T &value );

public:
  virtual typename LazyValue<T>::references_t references() const;

public:
  virtual T operator()( const typename LazyValue<T>::variables_t &
  variables ) const;

private:
  T value_;
};


/**
 * @brief Create lazy constant value.
 *
 * Create lazy constant value as described in the documentation of the
 * LazyConstant class. This templated helper function returns a smart
 * pointer to the newly created lazy value.
 *
 * @param value Constant lazy value.
 * @returns Pointer to new lazy value.
 */
template< typename T >
typename LazyValue<T>::ptr_t lazy_constant( const T &value );

#include "constant.ipp"

#endif //DICON_LAZY_VALUE_CONSTANT_HPP_
```

Listing B.84: src/lazy/value/constant.ipp

```
// -*- c++ -*-

/*---- [Distribution] ------------------------------------------------
 * This file is part of the Disease Control System DiCon.
 *
 * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
 * Designed and developed with the guidance of Nedialko B. Dimitrov
 * and Lauren Ancel Meyers at the University of Texas at Austin.
 *
 * DiCon is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * DiCon is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
 *-----------------------------------------------------------------*/

template< typename T >
inline
LazyConstant<T>::LazyConstant( const T &value )
  : value_(value)
{
}


template< typename T >
inline
typename LazyValue<T>::references_t
LazyConstant<T>::references() const {
  return typename LazyValue<T>::references_t();
}


template< typename T >
inline
T
LazyConstant<T>::operator()( const typename LazyValue<T>::variables_t
&variables ) const {
  return value_;
}


template< typename T >
inline
typename LazyValue<T>::ptr_t
lazy_constant( const T &value ) {
```

```
52    return typename LazyValue<T>::ptr_t( new LazyConstant<T>(value) );
53  }
```

Listing B.85: src/lazy/value/unary.hpp

```
1   #ifndef DICON_LAZY_VALUE_UNARY_HPP_
2   #define DICON_LAZY_VALUE_UNARY_HPP_
3
4   /*---- [Distribution] ------------------------------------------------
5    * This file is part of the Disease Control System DiCon.
6    *
7    * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8    * Designed and developed with the guidance of Nedialko B. Dimitrov
9    * and Lauren Ancel Meyers at the University of Texas at Austin.
10   *
11   * DiCon is free software: you can redistribute it and/or modify it
12   * under the terms of the GNU General Public License as published by
13   * the Free Software Foundation, either version 3 of the License, or
14   * (at your option) any later version.
15   *
16   * DiCon is distributed in the hope that it will be useful, but
17   * WITHOUT ANY WARRANTY; without even the implied warranty of
18   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19   * General Public License for more details.
20   *
21   * You should have received a copy of the GNU General Public License
22   * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23   *--------------------------------------------------------------------*/
24
25  /**
26   * @file
27   * @brief LazyUnaryOp class.
28   */
29  #include "../value.hpp"
30  #include <boost/noncopyable.hpp>
31
32
33  /**
34   * @brief Lazy unary operator.
35   *
36   * The LazyUnaryOp class implements the lazy application of a unary
37   * functor to a lazy value.
38   *
39   * The functor of type @e Op must be compatible with the following
40   * function type signature: <code>T(const T &)</code>.
41   *
42   * @tparam Op Functor to apply.
43   * @tparam T Lazy value's type.
44   */
45  template< typename Op, typename T = typename Op::result_type >
46  class LazyUnaryOp
47    : boost::noncopyable, public LazyValue<T>
48  {
49  public:
50    /**
51     * @brief Create lazy unary operator.
52     *
53     * Constructor that creates the lazy value that is the lazy
54     * application of the unary functor given by @e op to the lazy value
55     * given by @e x. The resulting lazy value will depend on all
56     * variables the original lazy value depends on.
57     *
58     * In order to fulfill the lazy value interface, the functor @e op
59     * must return the same value each time it is called with the same
60     * argument.
61     *
62     * @param x Original lazy value.
63     * @param op Functor to apply lazily.
64     */
65    LazyUnaryOp( typename LazyValue<T>::ptr_t &x, const Op &op = Op() );
66
67  public:
68    virtual typename LazyValue<T>::references_t references() const;
69
70  public:
71    virtual T operator()( const typename LazyValue<T>::variables_t &
      variables ) const;
72
73  private:
74    const typename LazyValue<T>::ptr_t x_;
75    Op op_;
76  };
77
78
79  /**
80   * @brief Create lazy unary operator.
81   *
82   * Create lazy unary operator as described in the documentation of the
83   * LazyUnaryOp class.  This templated helper function returns a smart
84   * pointer to the newly created lazy value.
85   *
86   * @param x Original lazy value.
87   * @param op Functor to apply lazily.
88   * @returns Pointer to new lazy value.
89   */
90  template< typename Op, typename T >
91  typename detail::LazyValuePtr<T>::ptr_t lazy_unary_op( T x, const Op &
      op );
92
93
94  #include "unary.ipp"
95
96  #endif //DICON_LAZY_VALUE_UNARY_HPP_
```

Listing B.86: src/lazy/value/unary.ipp

```
1   // -*- c++ -*-
2
3   /*---- [Distribution] ------------------------------------------------
4    * This file is part of the Disease Control System DiCon.
5    *
6    * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
7    * Designed and developed with the guidance of Nedialko B. Dimitrov
8    * and Lauren Ancel Meyers at the University of Texas at Austin.
9    *
10   * DiCon is free software: you can redistribute it and/or modify it
11   * under the terms of the GNU General Public License as published by
12   * the Free Software Foundation, either version 3 of the License, or
```

```
 * (at your option) any later version.
 *
 * DiCon is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
 *----------------------------------------------------------------------*/

template< typename Op, typename T >
inline
LazyUnaryOp<Op, T>::LazyUnaryOp( typename LazyValue<T>::ptr_t &x, const Op &op )
  : x_(x), op_(op)
{
}


template< typename Op, typename T >
inline
typename LazyValue<T>::references_t
LazyUnaryOp<Op, T>::references() const {
  return x_->references();
}


template< typename Op, typename T >
inline
T
LazyUnaryOp<Op, T>::operator()( const typename LazyValue<T>::variables_t &variables ) const {
  return op_( (*x_)(variables) );
}


template< typename Op, typename T >
inline
typename detail::LazyValuePtr<T>::ptr_t
lazy_unary_op( T x, const Op &op ) {
  return typename detail::LazyValuePtr<T>::ptr_t
    ( new LazyUnaryOp< Op
                     , typename detail::LazyValuePtr<T>::value_t
                     >(x, op)
    );
}
```

Listing B.87: src/lazy/value/variable.hpp

```
#ifndef DICON_LAZY_VALUE_VARIABLE_HPP_
#define DICON_LAZY_VALUE_VARIABLE_HPP_

/*---- [Distribution] ------------------------------------------------*
 * This file is part of the Disease Control System DiCon.
 *
 * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
 * Designed and developed with the guidance of Nedialko B. Dimitrov
 * and Lauren Ancel Meyers at the University of Texas at Austin.
 *
 * DiCon is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * DiCon is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
 *----------------------------------------------------------------------*/

/**
 * @file
 * @brief LazyVariable class.
 */
#include "../value.hpp"


/**
 * @brief Lazy variable value.
 *
 * The LazyVariable class implements the lazy value of a
 * variable. Thus, it is the root of all laziness in the LazyValue
 * hierarchy.
 *
 * @tparam T Lazy value's type.
 */
template< typename T >
class LazyVariable
  : public LazyValue<T>
{
public:
  /**
   * @brief Create lazy variable value.
   *
   * Constructor that creates the lazy value that evaluates to the
   * current assignment of the variable with name given by @e name;
   * the lazy value depends only on this variable.
   *
   * @param name Name of variable.
   */
  LazyVariable( const std::string &name );

public:
  virtual typename LazyValue<T>::references_t references() const;

public:
  virtual T operator()( const typename LazyValue<T>::variables_t &variables ) const;

private:
  std::string name_;
};


/**
 * @brief Create lazy variable value.
```

```
70   *
71   * Create lazy variable value as described in the documentation of the
72   * LazyVariable class. This templated helper function returns a smart
73   * pointer to the newly created lazy value.
74   *
75   * @param name Name of variable.
76   * @returns Pointer to new lazy value.
77   */
78  template< typename T >
79  typename LazyValue<T>::ptr_t lazy_variable( const std::string &name );
80
81
82  #include "variable.ipp"
83
84  #endif //DICON_LAZY_VALUE_VARIABLE_HPP_
```

Listing B.88: src/lazy/value/variable.ipp

```
1   // -*- c++ -*-
2
3   /*---- [Distribution] ------------------------------------------------
4    * This file is part of the Disease Control System DiCon.
5    *
6    * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
7    * Designed and developed with the guidance of Nedialko B. Dimitrov
8    * and Lauren Ancel Meyers at the University of Texas at Austin.
9    *
10   * DiCon is free software: you can redistribute it and/or modify it
11   * under the terms of the GNU General Public License as published by
12   * the Free Software Foundation, either version 3 of the License, or
13   * (at your option) any later version.
14   *
15   * DiCon is distributed in the hope that it will be useful, but
16   * WITHOUT ANY WARRANTY; without even the implied warranty of
17   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
18   * General Public License for more details.
19   *
20   * You should have received a copy of the GNU General Public License
21   * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
22   *--------------------------------------------------------------------*/
23
24  #include "../../error.hpp"
25  #include <boost/format.hpp>
26
27
28  template< typename T >
29  inline
30  LazyVariable<T>::LazyVariable( const std::string &name )
31    : name_(name)
32  {
33  }
34
35
36  template< typename T >
37  inline
38  typename LazyValue<T>::references_t
39  LazyVariable<T>::references() const {
40    typename LazyValue<T>::references_t references;
41
42    references.insert( name_ );
43
44    return references;
45  }
46
47
48  template< typename T >
49  inline
50  T
51  LazyVariable<T>::operator()( const typename LazyValue<T>::variables_t
    &variables ) const {
52    typename LazyValue<T>::variables_t::const_iterator it;
53
54    if( (it = variables.find(name_)) != variables.end() )
55      return it->second;
56    else
57      BOOST_THROW_EXCEPTION( AssertionError() << errinfo_value<std::
    string>(name_) );
58  }
59
60
61  template< typename T >
62  inline
63  typename LazyValue<T>::ptr_t
64  lazy_variable( const std::string &name ) {
65    return typename LazyValue<T>::ptr_t( new LazyVariable<T>(name) );
66  }
```

B.3 Optimizers

Listing B.89: optimizer/cpp/optimizer.hpp

```
#ifndef DICON_CPP_OPTIMIZER_HPP_
#define DICON_CPP_OPTIMIZER_HPP_

/*---- [Distribution] ------------------------------------------------
 * This file is part of the Disease Control System DiCon.
 *
 * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
 * Designed and developed with the guidance of Nedialko B. Dimitrov
 * and Lauren Ancel Meyers at the University of Texas at Austin.
 *
 * DiCon is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * DiCon is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
 *--------------------------------------------------------------------*/

#include <boost/noncopyable.hpp>
#include <boost/optional.hpp>
#include <stdexcept>
#include <string>
#include <vector>


namespace optimizer {

  typedef int node_t;
  typedef std::vector<node_t> policy_t;
  typedef std::vector<std::string> arguments_t;

  extern "C" typedef int (*children_t)( const node_t *path, const node_t **children );

}


extern "C" {

  int initialize( int argc, const char *const *argv,
                  optimizer::children_t children, const char *state_file     );
  int get_policy( const optimizer::node_t **policy                           );
  int update    ( const optimizer::node_t *policy, double reward             );
  int policies  ( unsigned *count,
                  const optimizer::node_t **policies, const double **rewards );
  int dump_state( const char *filename                                       );

  const char *get_error();

}


class OptimizerError
  : public std::runtime_error
{
public:
  OptimizerError( const std::string &what );
};


class Optimizer
  : boost::noncopyable
{
public:
  virtual ~Optimizer() {}

protected:
  std::vector<optimizer::node_t> children( const std::vector<optimizer::node_t> &path );

protected:
  typedef std::vector<std::pair<optimizer::policy_t, double> > policies_result_type;

  virtual boost::optional<optimizer::policy_t> get_policy() = 0;
  virtual void update( const optimizer::policy_t &policy, double reward ) = 0;
  virtual policies_result_type policies( unsigned count ) = 0;
  virtual void dump_state( const std::string &filename ) = 0;

private:
  friend int get_policy( const optimizer::node_t **                          );
  friend int update    ( const optimizer::node_t *, double                   );
  friend int policies  ( unsigned *,
                         const optimizer::node_t **, const double ** );
  friend int dump_state( const char *                                        );

  void get_policy_c( const optimizer::node_t **policy                        );
  void update_c    ( const optimizer::node_t *policy, double reward          );
  void policies_c  ( unsigned *count,
                     const optimizer::node_t **policies, const double **rewards );
  void dump_state_c( const char *filename                                    );

};

#endif
```

```
96  private:
97    optimizer::policy_t c_to_cpp( const optimizer::node_t *list );
98    const optimizer::node_t *cpp_to_c( const optimizer::policy_t &policy
    , bool keep = false );
99
100 private:
101   std::vector<optimizer::node_t> temporary_;
102   std::vector<double> tmp_rewards_;
103 };
104
105
106 extern
107 Optimizer *
108 create_optimizer( const std::string &name,
109                   const optimizer::arguments_t &arguments,
110                   const boost::optional<std::string> &state_file );
111
112 #define CREATE_OPTIMIZER( CLASSNAME )                                    
    \
113   Optimizer *create_optimizer( const std::string &name,                  
    \
114                                const optimizer::arguments_t &arguments    
    , \
115                                const boost::optional<std::string> &       
    state_file ) \
116   {                                                                      
    \
117     return new CLASSNAME ( name, arguments, state_file );                
    \
118   }                                                                      
    \
119   /**/
120
121 #endif //DICON_CPP_OPTIMIZER_HPP_
```

Listing B.90: optimizer/cpp/optimizer.cpp

```
1   /*--- [Distribution] ------------------------------------------------
2    * This file is part of the Disease Control System DiCon.
3    *
4    * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
5    * Designed and developed with the guidance of Nedialko B. Dimitrov
6    * and Lauren Ancel Meyers at the University of Texas at Austin.
7    *
8    * DiCon is free software: you can redistribute it and/or modify it
9    * under the terms of the GNU General Public License as published by
10   * the Free Software Foundation, either version 3 of the License, or
11   * (at your option) any later version.
12   *
13   * DiCon is distributed in the hope that it will be useful, but
14   * WITHOUT ANY WARRANTY; without even the implied warranty of
15   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
16   * General Public License for more details.
17   *
18   * You should have received a copy of the GNU General Public License
19   * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
20   *-------------------------------------------------------------------*/
21
22 #include "optimizer.hpp"
23 #include <boost/foreach.hpp>
24 #include <boost/scoped_ptr.hpp>
25
26
27 namespace detail {
28
29   static const char *c_error = NULL;
30   static std::string error_str = "No error.";
31   static optimizer::children_t children = NULL;
32   static boost::scoped_ptr<Optimizer> optimizer;
33
34
35   static
36   void
37   set_error() throw() {
38     c_error = "Unknown error.";
39   }
40
41
42   static
43   void
44   set_error( const std::string &str ) throw() {
45     try {
46       c_error = (error_str = str).c_str();
47     }
48     catch( ... ) {
49       set_error();
50     }
51   }
52
53 }
54
55
56
57 OptimizerError::OptimizerError( const std::string &what )
58   : std::runtime_error(what)
59 {
60 }
61
62
63 optimizer::policy_t
64 Optimizer::c_to_cpp( const optimizer::node_t *const list ) {
65   optimizer::policy_t policy;
66
67   for( size_t i = 0; list[i]; ++i )
68     policy.push_back( list[i] );
69
70   return policy;
71 }
72
73
74 const optimizer::node_t *
75 Optimizer::cpp_to_c( const optimizer::policy_t &policy, bool keep ) {
76   if( !keep )
77     temporary_.clear();
78
79   BOOST_FOREACH( optimizer::node_t node, policy ) {
80     if( node != 0 )
81       temporary_.push_back( node );
82     else
```

```
83        throw OptimizerError( "Optimizer::cpp_to_c(): Invalid node (0)." );
84      }
85
86      temporary_.push_back( 0 );
87      return temporary_.data();
88    }
89
90
91    std::vector<optimizer::node_t>
92    Optimizer::children( const std::vector<optimizer::node_t> &path ) {
93      if( !detail::children )
94        throw OptimizerError( "Optimizer::children(): No children callback set." );
95
96      const optimizer::node_t *children;
97
98      if( detail::children(cpp_to_c(path), &children) != 0 )
99        throw OptimizerError( "Optimizer::children(): Children callback failed." );
100
101     return c_to_cpp( children );
102   }
103
104
105   void
106   Optimizer::get_policy_c( const optimizer::node_t **const policy ) {
107     const boost::optional<optimizer::policy_t> &policy_cpp = get_policy();
108
109     if( policy_cpp )
110       *policy = cpp_to_c( *policy_cpp );
111     else
112       *policy = NULL;
113   }
114
115
116   void
117   Optimizer::update_c( const optimizer::node_t *const policy, double reward ) {
118     update( c_to_cpp(policy), reward );
119   }
120
121
122   void
123   Optimizer::policies_c( unsigned *const count, const optimizer::node_t **const policies, const double **rewards ) {
124     const policies_result_type &list = this->policies( *count );
125
126     *count = list.size();
127
128     temporary_.clear();
129     tmp_rewards_.clear();
130
131     BOOST_FOREACH( const policies_result_type::value_type &entry, list ) {
132       const optimizer::policy_t &policy = entry.first;
133       const double &reward = entry.second;
134
135       cpp_to_c( policy, true );
136       tmp_rewards_.push_back( reward );
137     }
138
139     *policies = temporary_.data();
140     *rewards = tmp_rewards_.data();
141   }
142
143
144   void
145   Optimizer::dump_state_c( const char *const filename ) {
146     dump_state( std::string(filename) );
147   }
148
149
150
151   extern "C" {
152
153     int
154     initialize( int argc, const char *const *const argv,
155                 optimizer::children_t children, const char *const state_file )
156     {
157       try {
158         detail::optimizer.reset();
159
160         std::string name;
161         optimizer::arguments_t arguments;
162
163         for( int i = 0; i < argc; ++i ) {
164           std::string arg( argv[i] );
165
166           if( i == 0 )
167             name = arg;
168           else
169             arguments.push_back( arg );
170         }
171
172         boost::optional<std::string> state_file_s;
173
174         if( state_file )
175           state_file_s = std::string( state_file );
176
177         detail::children = children;
178         detail::optimizer.reset( create_optimizer(name,
179                                                   arguments,
180                                                   state_file_s) );
181         return 0;
182       }
183       catch( std::exception &e ) {
184         detail::set_error( e.what() );
185         return -1;
186       }
187       catch( ... ) {
188         detail::set_error();
189         return -1;
190       }
191     }
192
193
194   #define CALL_OPTIMIZER_INSTANCE( NAME, ARGUMENTS ) \
```

```
195   try {                                                                    \
196     if( !detail::optimizer )                                               \
197       throw OptimizerError( #NAME "():_No_optimizer_instance." );          \
198                                                                            \
199     detail::optimizer->NAME##_c ARGUMENTS;                                 \
200     return 0;                                                              \
201   }                                                                        \
202   catch( std::exception &e ) {                                             \
203     detail::set_error( e.what() );                                         \
204     return -1;                                                             \
205   }                                                                        \
206   catch( ... ) {                                                           \
207     detail::set_error();                                                   \
208     return -1;                                                             \
209   }                                                                        \
210   /**/
211
212
213   int
214   get_policy( const optimizer::node_t **policy ) {
215 #define ARGUMENTS ( policy )
216     CALL_OPTIMIZER_INSTANCE( get_policy, ARGUMENTS )
217 #undef ARGUMENTS
218   }
219
220
221   int
222   update( const optimizer::node_t *policy, double reward ) {
223 #define ARGUMENTS ( policy, reward )
224     CALL_OPTIMIZER_INSTANCE( update, ARGUMENTS )
225 #undef ARGUMENTS
226   }
227
228
229   int
230   policies( unsigned *count, const optimizer::node_t **policies, const
      double **rewards ) {
231 #define ARGUMENTS ( count, policies, rewards )
232     CALL_OPTIMIZER_INSTANCE( policies, ARGUMENTS )
233 #undef ARGUMENTS
234   }
235
236
237   int
238   dump_state( const char *filename ) {
239 #define ARGUMENTS ( filename )
240     CALL_OPTIMIZER_INSTANCE( dump_state, ARGUMENTS )
241 #undef ARGUMENTS
242   }
243
244
245   const char *
246   get_error() {
247     if( !detail::c_error )
248       detail::set_error( "No_error." );
249
250     return detail::c_error;
251   }
252
253 }
```

Listing B.91: optimizer/cpp/exhaustive/exhaustive.hpp

```
1  #ifndef DICON_CPP_EXHAUSTIVE_EXHAUSTIVE_HPP_
2  #define DICON_CPP_EXHAUSTIVE_EXHAUSTIVE_HPP_
3
4  /*--- [Distribution] ------------------------------------------------------
5   * This file is part of the Disease Control System DiCon.
6   *
7   * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8   * Designed and developed with the guidance of Nedialko B. Dimitrov
9   * and Lauren Ancel Meyers at the University of Texas at Austin.
10  *
11  * DiCon is free software: you can redistribute it and/or modify it
12  * under the terms of the GNU General Public License as published by
13  * the Free Software Foundation, either version 3 of the License, or
14  * (at your option) any later version.
15  *
16  * DiCon is distributed in the hope that it will be useful, but
17  * WITHOUT ANY WARRANTY; without even the implied warranty of
18  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19  * General Public License for more details.
20  *
21  * You should have received a copy of the GNU General Public License
22  * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23  *---------------------------------------------------------------------*/
24
25 #include "../optimizer.hpp"
26 #include <boost/serialization/map.hpp>
27 #include <boost/serialization/vector.hpp>
28
29
30 class ExhaustiveSearch
31   : public Optimizer
32 {
33 public:
34   ExhaustiveSearch( const std::string &name,
35                     const optimizer::arguments_t &arguments,
36                     const boost::optional<std::string> &state_file );
37
38 protected:
39   virtual boost::optional<optimizer::policy_t> get_policy();
40   virtual void update( const optimizer::policy_t &policy, double
      reward );
41   virtual policies_result_type policies( unsigned count );
```

```
42    virtual void dump_state( const std::string &filename );
43
44  private:
45    void load_state( std::istream &in );
46    void save_state( std::ostream &out );
47
48  private:
49    typedef std::multimap<double, optimizer::policy_t> best_policies_t;
50
51    struct Data {
52      size_t max_best_policies;
53      best_policies_t best_policies;
54      std::vector<optimizer::policy_t> queue;
55    };
56
57    Data data;
58  };
59
60
61  namespace boost {
62    namespace serialization {
63
64      template< class Archive >
65      void serialize( Archive &ar, ExhaustiveSearch::Data &data, const
    unsigned int version );
66
67    }
68  }
69
70
71  #include "exhaustive.ipp"
72
73  #endif //DICON_CPP_EXHAUSTIVE_EXHAUSTIVE_HPP_
```

Listing B.92: optimizer/cpp/exhaustive/exhaustive.ipp

```
1   // -*- c++ -*-
2
3   /*---- [Distribution] ------------------------------------------------
4    * This file is part of the Disease Control System DiCon.
5    *
6    * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
7    * Designed and developed with the guidance of Nedialko B. Dimitrov
8    * and Lauren Ancel Meyers at the University of Texas at Austin.
9    *
10   * DiCon is free software: you  can redistribute it and/or modify it
11   * under the terms of the GNU General Public License as published by
12   * the Free Software Foundation, either version 3 of the License, or
13   * (at your option) any later version.
14   *
15   * DiCon  is distributed in  the hope  that it  will be  useful, but
16   * WITHOUT ANY WARRANTY;  without  even the  implied  warranty  of
17   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
18   * General Public License for more details.
19   *
20   * You should have received a copy of the GNU General Public License
21   * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
22   *--------------------------------------------------------------------*/
23
24  namespace boost {
25    namespace serialization {
26
27      template< class Archive >
28      inline
29      void
30      serialize( Archive &ar, ExhaustiveSearch::Data &data, const
    unsigned int version ) {
31        ar & boost::serialization::make_nvp( "max_best_policies", data.
    max_best_policies );
32        ar & boost::serialization::make_nvp( "best_policies"    , data.
    best_policies     );
33        ar & boost::serialization::make_nvp( "queue"            , data.
    queue             );
34      }
35
36    }
37  }
```

Listing B.93: optimizer/cpp/exhaustive/exhaustive.cpp

```
1   /*---- [Distribution] ------------------------------------------------
2    * This file is part of the Disease Control System DiCon.
3    *
4    * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
5    * Designed and developed with the guidance of Nedialko B. Dimitrov
6    * and Lauren Ancel Meyers at the University of Texas at Austin.
7    *
8    * DiCon is free software: you  can redistribute it and/or modify it
9    * under the terms of the GNU General Public License as published by
10   * the Free Software Foundation, either version 3 of the License, or
11   * (at your option) any later version.
12   *
13   * DiCon  is distributed in  the hope  that it  will be  useful, but
14   * WITHOUT ANY WARRANTY;  without  even the  implied  warranty  of
15   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
16   * General Public License for more details.
17   *
18   * You should have received a copy of the GNU General Public License
19   * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
20   *--------------------------------------------------------------------*/
21
22  #include "exhaustive.hpp"
23  #include <boost/archive/binary_iarchive.hpp>
24  #include <boost/archive/binary_oarchive.hpp>
25  #include <boost/foreach.hpp>
26  #include <boost/lexical_cast.hpp>
27  #include <fstream>
28  #include <limits>
29
30
31  static const size_t default_max_best_policies = 10;
32
33
34  ExhaustiveSearch::ExhaustiveSearch( const std::string &name,
35                                      const optimizer::arguments_t &
    arguments,
36                                      const boost::optional<std::string>
    &state_file )
```

```
{
  if( !state_file ) {
    // Initialize new optimizer.
    data.queue.push_back( optimizer::policy_t() );
    data.max_best_policies = default_max_best_policies;

    if( !arguments.empty() ) {
      if( arguments.size() == 2 && arguments[0] == "-n" )
        data.max_best_policies = boost::lexical_cast<size_t>( arguments[1] );
      else if( arguments.size() == 1 && arguments[0].substr(0, 2) == "-n" )
        data.max_best_policies = boost::lexical_cast<size_t>( arguments[0].substr(2) );
      else
        throw OptimizerError( "ExhaustiveSearch::ExhaustiveSearch():_Failed_to_parse_arguments." );
    }
  }
  else {
    // Resume: get state from given file.
    std::ifstream in( state_file->c_str() );
    load_state( in );
    if( in.fail() )
      throw OptimizerError( "ExhaustiveSearch::ExhaustiveSearch():_Failed_to_read_state_file." );
  }
}


boost::optional<optimizer::policy_t>
ExhaustiveSearch::get_policy() {
  while( true ) {
    if( data.queue.empty() )
      return boost::none;

    optimizer::policy_t path = data.queue.back();
    data.queue.pop_back();

    const optimizer::policy_t &list = children( path );

    if( list.empty() ) {
      // Reached leaf.
      return path;
    }
    else {
      for( optimizer::policy_t::const_reverse_iterator it = list.rbegin(); it != list.rend(); ++it ) {
        optimizer::policy_t new_path = path;
        new_path.push_back( *it );
        data.queue.push_back( new_path );
      }
    }
  }
}


void
ExhaustiveSearch::update( const optimizer::policy_t &policy, double reward ) {
  data.best_policies.insert( std::make_pair(reward, policy) );

  if( data.best_policies.size() > data.max_best_policies ) {
    assert( !data.best_policies.empty() );
    data.best_policies.erase( data.best_policies.begin() );
  }
}


ExhaustiveSearch::policies_result_type
ExhaustiveSearch::policies( unsigned count ) {
  policies_result_type result;

  for( best_policies_t::const_reverse_iterator it = data.best_policies.rbegin(); it != data.best_policies.rend(); ++it ) {
    result.push_back( policies_result_type::value_type(it->second, it->first) );
  }

  return result;
}


void
ExhaustiveSearch::dump_state( const std::string &filename ) {
  std::ofstream out( filename.c_str() );
  save_state( out );
  if( out.fail() )
    throw OptimizerError( "ExhaustiveSearch::dump_state():_Failed_to_write_to_file." );
}


void
ExhaustiveSearch::load_state( std::istream &in ) {
  boost::archive::binary_iarchive( in )
    >> boost::serialization::make_nvp( "data", data );
}


void
ExhaustiveSearch::save_state( std::ostream &out ) {
  boost::archive::binary_oarchive( out )
    << boost::serialization::make_nvp( "data", data );
}
```

Listing B.94: optimizer/cpp/exhaustive/main.cpp

```
/*---- [Distribution] ------------------------------------------------
 * This file is part of the Disease Control System DiCon.
 *
 * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
 * Designed and developed with the guidance of Nedialko B. Dimitrov
 * and Lauren Ancel Meyers at the University of Texas at Austin.
 *
 * DiCon is free software: you  can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
```

```
 * DiCon is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
 *-------------------------------------------------------------------*/

#include "exhaustive.hpp"

CREATE_OPTIMIZER( ExhaustiveSearch )
```

Listing B.95: optimizer/cpp/uct/main.cpp

```
/*---- [Distribution] ---------------------------------------------------
 * This file is part of the Disease Control System DiCon.
 *
 * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
 * Designed and developed with the guidance of Nedialko B. Dimitrov
 * and Lauren Ancel Meyers at the University of Texas at Austin.
 *
 * DiCon is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * DiCon is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
 *-------------------------------------------------------------------*/

#include "uct.hpp"

CREATE_OPTIMIZER( UCT )
```

Listing B.96: optimizer/cpp/uct/uct.hpp

```
#ifndef DICON_CPP_UCT_UCT_HPP_
#define DICON_CPP_UCT_UCT_HPP_

/*---- [Distribution] ---------------------------------------------------
 * This file is part of the Disease Control System DiCon.
 *
 * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
 * Designed and developed with the guidance of Nedialko B. Dimitrov
 * and Lauren Ancel Meyers at the University of Texas at Austin.
 *
 * DiCon is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * DiCon is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
 *-------------------------------------------------------------------*/

#include "../optimizer.hpp"
#include <boost/variant.hpp>
#include <boost/serialization/map.hpp>
#include <boost/serialization/vector.hpp>


class UCT
  : public Optimizer
{
public:
  UCT( const std::string &name,
       const optimizer::arguments_t &arguments,
       const boost::optional<std::string> &state_file );

protected:
  virtual boost::optional<optimizer::policy_t> get_policy();
  virtual void update( const optimizer::policy_t &policy, double reward );
  virtual policies_result_type policies( unsigned count );
  virtual void dump_state( const std::string &filename );

private:
  void load_state( std::istream &in );
  void save_state( std::ostream &out );

public:
  struct Node;

  typedef std::map<optimizer::node_t, Node> tree_t;

  struct Node {
    Node();

    bool is_leaf;
    double reward;
    unsigned count;
    tree_t children;
  };

private:
  struct Data {
    Node root_node;
    std::vector<Node *> current_path;
  };

  Data data;
};


BOOST_SERIALIZATION_SPLIT_FREE( UCT::Data )

```

```
75  namespace boost {
76    namespace serialization {
77
78      template< class Archive >
79      void save( Archive &ar, const UCT::Data &data, const unsigned int
    version );
80
81      template< class Archive >
82      void load( Archive &ar, UCT::Data &data, const unsigned int
    version );
83
84    }
85  }
86
87
88  namespace boost {
89    namespace serialization {
90
91      template< class Archive >
92      void serialize( Archive &ar, UCT::Node &node, const unsigned int
    version );
93
94    }
95  }
96
97
98  #include "uct.ipp"
99
100 #endif //DICON_CPP_UCT_UCT_HPP_
```

Listing B.97: optimizer/cpp/uct/uct.ipp

```
1   // -*- c++ -*-
2
3   /*---- [Distribution] ------------------------------------------------
4    * This file is part of the Disease Control System DiCon.
5    *
6    * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
7    * Designed and developed with the guidance of Nedialko B. Dimitrov
8    * and Lauren Ancel Meyers at the University of Texas at Austin.
9    *
10   * DiCon is free software: you can redistribute it and/or modify it
11   * under the terms of the GNU General Public License as published by
12   * the Free Software Foundation, either version 3 of the License, or
13   * (at your option) any later version.
14   *
15   * DiCon is distributed in the hope that it will be useful, but
16   * WITHOUT ANY WARRANTY; without even the implied warranty of
17   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
18   * General Public License for more details.
19   *
20   * You should have received a copy of the GNU General Public License
21   * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
22   *--------------------------------------------------------------------*/
23
24  namespace boost {
25    namespace serialization {
26
27      template< class Archive >
28      inline
29      void
30      save( Archive &ar, const UCT::Data &data, const unsigned int
    version ) {
31        if( !data.current_path.empty() )
32          throw OptimizerError( "save():_Policy_pending." );
33
34        ar & boost::serialization::make_nvp( "root_node", data.root_node
    );
35      }
36
37      template< class Archive >
38      inline
39      void
40      load( Archive &ar, UCT::Data &data, const unsigned int version ) {
41        ar & boost::serialization::make_nvp( "root_node", data.root_node
    );
42      }
43
44    }
45  }
46
47
48  namespace boost {
49    namespace serialization {
50
51      template< class Archive >
52      inline
53      void
54      serialize( Archive &ar, UCT::Node &node, const unsigned int
    version ) {
55        ar & boost::serialization::make_nvp( "is_leaf" , node.is_leaf   )
    ;
56        ar & boost::serialization::make_nvp( "reward"  , node.reward    )
    ;
57        ar & boost::serialization::make_nvp( "count"   , node.count     )
    ;
58        ar & boost::serialization::make_nvp( "children", node.children )
    ;
59      }
60
61    }
62  }
```

Listing B.98: optimizer/cpp/uct/uct.cpp

```
1   /*---- [Distribution] ------------------------------------------------
2    * This file is part of the Disease Control System DiCon.
3    *
4    * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
5    * Designed and developed with the guidance of Nedialko B. Dimitrov
6    * and Lauren Ancel Meyers at the University of Texas at Austin.
7    *
8    * DiCon is free software: you can redistribute it and/or modify it
9    * under the terms of the GNU General Public License as published by
10   * the Free Software Foundation, either version 3 of the License, or
11   * (at your option) any later version.
12   *
13   * DiCon is distributed in the hope that it will be useful, but
```

```
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
 *----------------------------------------------------------------------*/

#include "uct.hpp"
#include <boost/archive/binary_iarchive.hpp>
#include <boost/archive/binary_oarchive.hpp>
#include <boost/foreach.hpp>
#include <boost/format.hpp>
#include <boost/tuple/tuple.hpp>
#include <cmath>
#include <fstream>
#include <limits>


UCT::Node::Node()
  : is_leaf(false), reward(0), count(0)
{
}


UCT::UCT( const std::string &name,
          const optimizer::arguments_t &arguments,
          const boost::optional<std::string> &state_file )
{
  if( state_file ) {
    // Resume: get state from given file.
    std::ifstream in( state_file->c_str() );
    load_state( in );
    if( in.fail() )
      throw OptimizerError( "UCT::UCT():_Failed_to_read_state_file." );
  }
}


template< bool weighted >
static
std::pair<optimizer::node_t, UCT::Node *>
select_child( UCT::Node *const parent ) {
  assert( !parent->is_leaf );
  assert( !parent->children.empty() );

  UCT::Node *best_child = NULL;
  optimizer::node_t best_id = optimizer::node_t();
  double best_value = -std::numeric_limits<double>::infinity();

  BOOST_FOREACH( UCT::tree_t::value_type &entry, parent->children ) {
    UCT::Node *const child = &entry.second;

    if( weighted && child->count == 0 ) {
      best_id = entry.first;
      best_child = child;
      break;
    }

    if( child->count == 0 )
      continue;

    assert( child->count != 0 );
    assert( parent->count != 0 );

    double value = weighted
                   ? child->reward + std::sqrt( std::log(parent->count) * 2 / child->count )
                   : child->reward;

    if( !std::isfinite(value) )
      throw OptimizerError( (boost::format("select_child():_Invalid_weight_(%f).") % value).str() );

    if( value > best_value ) {
      best_id = entry.first;
      best_child = child;
      best_value = value;
    }
  }

  assert( best_child );
  return std::make_pair( best_id, best_child );
}


boost::optional<optimizer::policy_t>
UCT::get_policy() {
  if( !data.current_path.empty() ) {
    // One policy at a time.
    return boost::none;
  }

  optimizer::policy_t policy;

  assert( data.current_path.empty() );

  for( Node *node = &data.root_node;; ) {
    data.current_path.push_back( node );

    if( node->is_leaf )
      break;

    if( node->children.empty() ) {
      BOOST_FOREACH( optimizer::node_t child, children(policy) ) {
        // Create child in tree.
        node->children[child];
      }

      if( node->children.empty() ) {
        node->is_leaf = true;
        break;
      }
    }

    optimizer::node_t id;
    boost::tie( id, node )
      = select_child<true>( node );

    policy.push_back( id );
  }
```

```
133
134   return policy;
135 }
136
137
138 void
139 UCT::update( const optimizer::policy_t &policy, double reward ) {
140   if( data.current_path.empty() )
141     throw OptimizerError( "UCT::update():_Unexpected_update." );
142
143   BOOST_FOREACH( Node *node, data.current_path )
144     node->reward += (reward - node->reward) / ++node->count;
145
146   data.current_path.clear();
147 }
148
149
150 UCT::policies_result_type
151 UCT::policies( unsigned count ) {
152   if( count == 0 )
153     return policies_result_type();
154
155   if( !data.current_path.empty() )
156     throw OptimizerError( "UCT::policies():_Update_pending." );
157
158   optimizer::policy_t policy;
159
160   Node *node;
161   for( node = &data.root_node; !node->is_leaf; ) {
162     if( node->children.empty() ) {
163       // Update has not been called.
164       return policies_result_type();
165     }
166
167     optimizer::node_t id;
168     boost::tie( id, node )
169       = select_child<false>( node );
170
171     policy.push_back( id );
172   }
173
174   assert( node->count != 0 );
175   return policies_result_type( 1, policies_result_type::value_type(
    policy, node->reward) );
176 }
177
178
179 void
180 UCT::dump_state( const std::string &filename ) {
181   std::ofstream out( filename.c_str() );
182   save_state( out );
183   if( out.fail() )
184     throw OptimizerError( "UCT::dump_state():_Failed_to_write_to_file.
    " );
185 }
186
187
188 void
189 UCT::load_state( std::istream &in ) {
190   boost::archive::binary_iarchive( in )
191     >> boost::serialization::make_nvp( "data", data );
192 }
193
194
195 void
196 UCT::save_state( std::ostream &out ) {
197   boost::archive::binary_oarchive( out )
198     << boost::serialization::make_nvp( "data", data );
199 }
```

B.4 Simulators

Listing B.99: simulator/cpp/communicator.hpp

```
#ifndef DICON_CPP_COMMUNICATOR_HPP_
#define DICON_CPP_COMMUNICATOR_HPP_

/*---- [Distribution] ------------------------------------------------
 * This file is part of the Disease Control System DiCon.
 *
 * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
 * Designed and developed with the guidance of Nedialko B. Dimitrov
 * and Lauren Ancel Meyers at the University of Texas at Austin.
 *
 * DiCon is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * DiCon is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
 *--------------------------------------------------------------------*/

#include <boost/noncopyable.hpp>
#include <stdexcept>
#include <stdint.h>


class CommunicatorError
  : public std::runtime_error
{
public:
  CommunicatorError( const std::string &what );
};


class Communicator
  : boost::noncopyable
{
public:
  Communicator( int input, int output );
  virtual ~Communicator();

  std::string read_message();
  void write_message( const std::string &msg );

private:
  std::string read_n( size_t n );
  void write( const std::string &buffer );
  void flush();

private:
  std::string encode_uint32( uint32_t value );
  uint32_t decode_uint32( const std::string &str );

private:
  int input_;
  int output_;
};

#endif //DICON_CPP_COMMUNICATOR_HPP_
```

Listing B.100: simulator/cpp/communicator.cpp

```
/*---- [Distribution] ------------------------------------------------
 * This file is part of the Disease Control System DiCon.
 *
 * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
 * Designed and developed with the guidance of Nedialko B. Dimitrov
 * and Lauren Ancel Meyers at the University of Texas at Austin.
 *
 * DiCon is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * DiCon is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
 *--------------------------------------------------------------------*/

#include "communicator.hpp"
#include <boost/scoped_array.hpp>
#include <cerrno>


CommunicatorError::CommunicatorError( const std::string &what )
  : std::runtime_error(what)
{
}


Communicator::Communicator( int input, int output )
  : input_(input), output_(output)
{
}


Communicator::~Communicator() {
  close( input_ );
  close( output_ );
  // Ignore errors.
}
```

```
46 std::string
47 Communicator::read_n( size_t n ) {
48   boost::scoped_array<char> buffer( new char[n] );
49
50   size_t done = 0;
51   while( done < n ) {
52     ssize_t res;
53     if( (res = ::read(input_, buffer.get()+done, n-done)) <= 0 )
54       throw CommunicatorError( "Communicator::read_n():_Failed_to_read
   _from_input." );
55
56     assert( size_t(res) <= n-done );
57
58     done += size_t(res);
59   }
60
61   return std::string( buffer.get(), n );
62 }
63
64
65 void
66 Communicator::write( const std::string &buffer ) {
67   const size_t n = buffer.size();
68
69   size_t done = 0;
70   while( done < n ) {
71     ssize_t res;
72     if( (res = ::write(output_, buffer.c_str()+done, n-done)) <= 0 )
73       throw CommunicatorError( "Communicator::write():_Failed_to_write
   _to_output." );
74
75     assert( size_t(res) <= n-done );
76
77     done += size_t(res);
78   }
79 }
80
81
82 void
83 Communicator::flush() {
84   if( ::fsync(output_) != 0 && errno != EINVAL )
85     throw CommunicatorError( "Communicator::flush():_Failed_to_flush_
   output." );
86 }
87
88
89 std::string
90 Communicator::encode_uint32( uint32_t value ) {
91   std::string msg( 4, char(0) );
92
93   msg[0] = char(static_cast<unsigned char>((value >>  0) & 255));
94   msg[1] = char(static_cast<unsigned char>((value >>  8) & 255));
95   msg[2] = char(static_cast<unsigned char>((value >> 16) & 255));
96   msg[3] = char(static_cast<unsigned char>((value >> 24) & 255));
97
98   return msg;
99 }
100
101
102 uint32_t
103 Communicator::decode_uint32( const std::string &str ) {
104   if( str.size() != 4 )
105     throw CommunicatorError( "Communicator::decode_uint32():_String_
   has_unexpected_length." );
106
107   return
108     (uint32_t(static_cast<unsigned char>(str[0])) <<  0) |
109     (uint32_t(static_cast<unsigned char>(str[1])) <<  8) |
110     (uint32_t(static_cast<unsigned char>(str[2])) << 16) |
111     (uint32_t(static_cast<unsigned char>(str[3])) << 24);
112 }
113
114
115 std::string
116 Communicator::read_message() {
117   size_t length = decode_uint32( read_n(4) );
118   return read_n( length );
119 }
120
121
122 void
123 Communicator::write_message( const std::string &msg ) {
124   write( encode_uint32(msg.length()) );
125   write( msg );
126   flush();
127 }
```

Listing B.101: simulator/cpp/simulator.hpp

```
1  #ifndef DICON_CPP_SIMULATOR_HPP_
2  #define DICON_CPP_SIMULATOR_HPP_
3
4  /*--- [Distribution] ---------------------------------------------------
5   * This file is part of the Disease Control System DiCon.
6   *
7   * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
8   * Designed and developed with the guidance of Nedialko B. Dimitrov
9   * and Lauren Ancel Meyers at the University of Texas at Austin.
10  *
11  * DiCon is free software: you can redistribute it and/or modify it
12  * under the terms of the GNU General Public License as published by
13  * the Free Software Foundation, either version 3 of the License, or
14  * (at your option) any later version.
15  *
16  * DiCon is distributed in the hope that it will be useful, but
17  * WITHOUT ANY WARRANTY; without even the implied warranty of
18  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
19  * General Public License for more details.
20  *
21  * You should have received a copy of the GNU General Public License
22  * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
23  *-----------------------------------------------------------------------*/
24
25 #include "communicator.hpp"
26 #include "simulator.pb.h"
27 #include <vector>
28
29
30 template< typename T >
31 class Simulator
```

```
32    : public Communicator
33  {
34  public:
35    Simulator();
36
37  public:
38    void main();
39
40  public:
41    virtual std::vector<T> children( const std::vector<T> &path ) = 0;
42    virtual double simulate( const std::vector<T> &policy ) = 0;
43    virtual std::string display( const std::vector<T> &policy ) = 0;
44
45  private:
46    proto::simulator::Question read_question();
47    proto::simulator::Answer new_answer();
48    void write_answer( const proto::simulator::Answer &answer );
49
50  private:
51    std::string pickle( const T &obj );
52    T unpickle( const std::string &str );
53
54  private:
55    void process();
56  };
57
58
59  #include "simulator.ipp"
60
61  #endif //DICON_CPP_SIMULATOR_HPP_
```

Listing B.102: simulator/cpp/simulator.ipp

```
1   // -*- c++ -*-
2
3   /*---- [Distribution] ------------------------------------------------
4    * This file is part of the Disease Control System DiCon.
5    *
6    * Copyright (C) 2009 Sebastian Goll, University of Texas at Austin
7    * Designed and developed with the guidance of Nedialko B. Dimitrov
8    * and Lauren Ancel Meyers at the University of Texas at Austin.
9    *
10   * DiCon is free software: you can redistribute it and/or modify it
11   * under the terms of the GNU General Public License as published by
12   * the Free Software Foundation, either version 3 of the License, or
13   * (at your option) any later version.
14   *
15   * DiCon is distributed in the hope that it will be useful, but
16   * WITHOUT ANY WARRANTY; without even the implied warranty of
17   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
18   * General Public License for more details.
19   *
20   * You should have received a copy of the GNU General Public License
21   * along with DiCon. If not, see <http://www.gnu.org/licenses/>.
22   *-------------------------------------------------------------------*/
23
24  #include <boost/archive/binary_iarchive.hpp>
25  #include <boost/archive/binary_oarchive.hpp>
26  #include <boost/exception.hpp>
27  #include <boost/foreach.hpp>
28  #include <boost/optional.hpp>
29  #include <boost/version.hpp>
30  #include <sstream>
31
32
33  template< typename T >
34  inline
35  Simulator<T>::Simulator()
36    : Communicator(STDIN_FILENO, STDOUT_FILENO)
37  {
38    GOOGLE_PROTOBUF_VERIFY_VERSION;
39  }
40
41
42  template< typename T >
43  inline
44  proto::simulator::Question
45  Simulator<T>::read_question() {
46    proto::simulator::Question question;
47    question.ParseFromString( read_message() );
48    return question;
49  }
50
51
52  template< typename T >
53  inline
54  proto::simulator::Answer
55  Simulator<T>::new_answer() {
56    proto::simulator::Answer answer;
57    answer.set_failed( false );
58    return answer;
59  }
60
61
62  template< typename T >
63  inline
64  void
65  Simulator<T>::write_answer( const proto::simulator::Answer &answer ) {
66    std::string str;
67    answer.SerializeToString( &str );
68    write_message( str );
69  }
70
71
72  template< typename T >
73  inline
74  std::string
75  Simulator<T>::pickle( const T &obj ) {
76    std::stringstream out;
77    {
78      boost::archive::binary_oarchive oa( out );
79      oa << obj;
80    }
81    return out.str();
82  }
83
84
85  template< typename T >
86  inline
87  T
```

```
88  Simulator<T>::unpickle( const std::string &str ) {
89    std::stringstream in( str );
90    T obj;
91    {
92      boost::archive::binary_iarchive ia( in );
93      ia >> obj;
94    }
95    return obj;
96  }
97
98
99  template< typename T >
100 inline
101 void
102 Simulator<T>::process() {
103   proto::simulator::Answer answer = new_answer();
104   proto::simulator::Question question = read_question();
105
106   boost::optional<std::string> failure;
107
108   try {
109     if( question.type() == proto::simulator::SIMULATE ) {
110       std::vector<T> policy( question.q_simulate().node_size() );
111       for( size_t i = 0; i < policy.size(); ++i )
112         policy[i] = unpickle( question.q_simulate().node(i) );
113       answer.mutable_a_simulate()->set_reward( simulate(policy) );
114     }
115     else if( question.type() == proto::simulator::CHILDREN ) {
116       std::vector<T> path( question.q_children().node_size() );
117       for( size_t i = 0; i < path.size(); ++i )
118         path[i] = unpickle( question.q_children().node(i) );
119       BOOST_FOREACH( const T &child, children(path) )
120         answer.mutable_a_children()->add_node( pickle(child) );
121     }
122     else if( question.type() == proto::simulator::DISPLAY ) {
123       std::vector<T> policy( question.q_display().node_size() );
124       for( size_t i = 0; i < policy.size(); ++i )
125         policy[i] = unpickle( question.q_display().node(i) );
126       answer.mutable_a_display()->set_display( display(policy) );
127     }
128   }
129 #if BOOST_VERSION >= 103900
130   catch( ... ) {
131     failure = boost::current_exception_diagnostic_information();
132   }
133 #else
134   catch( boost::exception &e ) {
135     failure = boost::diagnostic_information( e );
136   }
137   catch( std::exception &e ) {
138     failure = e.what();
139   }
140   catch( ... ) {
141     failure = "<unknown_exception>\n";
142   }
143 #endif
144
145   if( failure ) {
146     answer.set_failed( true );
147     answer.mutable_a_failure()->set_what( *failure );
148   }
149
150   write_answer( answer );
151 }
152
153
154 template< typename T >
155 inline
156 void
157 Simulator<T>::main() {
158   while( true )
159     process();
160 }
```

Listing B.103: simulator/cpp/simple_simulator/main.cpp

```
1   /*--- [Distribution] ---------------------------------------------------
2    * This file is part of the Disease Control System DiCon.
3    *
4    * Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
5    * Designed and developed with the guidance of Nedialko B. Dimitrov
6    * and Lauren Ancel Meyers at the University of Texas at Austin.
7    *
8    * DiCon is free software: you can redistribute it and/or modify it
9    * under the terms of the GNU General Public License as published by
10   * the Free Software Foundation, either version 3 of the License, or
11   * (at your option) any later version.
12   *
13   * DiCon is distributed in the hope that it will be useful, but
14   * WITHOUT ANY WARRANTY; without even the implied warranty of
15   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
16   * General Public License for more details.
17   *
18   * You should have received a copy of the GNU General Public License
19   * along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
20   *-----------------------------------------------------------------*/
21
22  #include "../simulator.hpp"
23  #include <boost/lexical_cast.hpp>
24
25
26  static size_t base = 5;
27  static size_t depth = 2;
28  typedef unsigned node_t;
29
30
31  class SimpleSimulator
32    : public Simulator<node_t>
33  {
34  public:
35    virtual
36    std::vector<node_t>
37    children( const std::vector<node_t> &path ) {
38      std::vector<node_t> res;
39
40      if( path.size() < depth ) {
41        for( size_t i = 0; i < base; ++i )
42          res.push_back( i );
43      }
44
```

```
45      return res;
46    }
47
48
49    virtual
50    double
51    simulate( const std::vector<node_t> &policy ) {
52      double value = 0;
53      double factor = 1;
54
55      BOOST_FOREACH( const node_t &digit, policy ) {
56        factor /= base;
57        value += digit * factor;
58      }
59
60      return value;
61    }
62
63
64    virtual
65    std::string
66    display( const std::vector<node_t> &policy ) {
67      std::string res;
68
69      BOOST_FOREACH( const node_t &digit, policy ) {
70        if( !res.empty() ) res += ",";
71        res += boost::lexical_cast<std::string>( digit );
72      }
73
74      return '[' + res + ']';
75    }
76  };
77
78
79  int
80  main() {
81    SimpleSimulator().main();
82  }
```

Listing B.104: simulator/simple_simulator.py

```
1  #!/usr/bin/env python
2
3  #--- [Distribution] ----------------------------------------------------
4  # This file is part of the Disease Control System DiCon.
5  #
6  # Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
7  # Designed and developed with the guidance of Nedialko B. Dimitrov
8  # and Lauren Ancel Meyers at the University of Texas at Austin.
9  #
10 # DiCon is free software: you can redistribute it and/or modify it
11 # under the terms of the GNU General Public License as published by
12 # the Free Software Foundation, either version 3 of the License, or
13 # (at your option) any later version.
14 #
15 # DiCon is distributed in the hope that it will be useful, but
16 # WITHOUT ANY WARRANTY; without even the implied warranty of
17 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
18 # General Public License for more details.
19 #
20 # You should have received a copy of the GNU General Public License
21 # along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
22 #-----------------------------------------------------------------------
23
24 from python.simulator import Simulator
25
26
27 base = 5
28 depth = 2
29
30
31 class SimpleSimulator( Simulator ):
32     def children( self, path ):
33         if len(path) < depth:
34             return range( 0, base )
35         else:
36             return []
37
38     def simulate( self, policy ):
39         value = 0.0
40         factor = 1.0
41
42         for digit in policy:
43             factor /= base
44             value += digit * factor
45
46         return value
47
48     def display( self, policy ):
49         return str( policy )
50
51
52 if __name__ == "__main__":
53     SimpleSimulator().main()
```

Listing B.105: simulator/python/__init__.py

```
1  #!/usr/bin/env python
2
3  #--- [Distribution] ----------------------------------------------------
4  # This file is part of the Disease Control System DiCon.
5  #
6  # Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
7  # Designed and developed with the guidance of Nedialko B. Dimitrov
8  # and Lauren Ancel Meyers at the University of Texas at Austin.
9  #
10 # DiCon is free software: you can redistribute it and/or modify it
11 # under the terms of the GNU General Public License as published by
12 # the Free Software Foundation, either version 3 of the License, or
13 # (at your option) any later version.
14 #
15 # DiCon is distributed in the hope that it will be useful, but
16 # WITHOUT ANY WARRANTY; without even the implied warranty of
17 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
18 # General Public License for more details.
19 #
20 # You should have received a copy of the GNU General Public License
21 # along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
```

```
#-----------------------------------------------------------------------
```

Listing B.106: simulator/python/communicator.py

```
#!/usr/bin/env python

#--- [Distribution] ----------------------------------------------------
# This file is part of the Disease Control System DiCon.
#
# Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
# Designed and developed with the guidance of Nedialko B. Dimitrov
# and Lauren Ancel Meyers at the University of Texas at Austin.
#
# DiCon is free software: you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# DiCon is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
#-----------------------------------------------------------------------

class Communicator:
    def __init__( self, input, output ):
        self.input = input
        self.output = output


    def read_n( self, n ):
        buffer = ""

        while len(buffer) < n:
            res = self.input.read( n-len(buffer) )

            if len(res) == 0:
                raise IOError, "Failed_to_read_from_stream."

            buffer += res

        return buffer


    def write( self, buffer ):
        self.output.write( buffer )


    def flush( self ):
        self.output.flush()


    def encode_uint32( self, value ):
        return \
            chr((value >>  0) & 255) + \
            chr((value >>  8) & 255) + \
            chr((value >> 16) & 255) + \
            chr((value >> 24) & 255)


    def decode_uint32( self, str ):
        if len(str) != 4:
            raise ValueError, "String_has_unexpected_length."

        return \
            (ord(str[0]) <<  0) | \
            (ord(str[1]) <<  8) | \
            (ord(str[2]) << 16) | \
            (ord(str[3]) << 24)


    def read_message( self ):
        length = self.decode_uint32( self.read_n(4) )
        return self.read_n( length )


    def write_message( self, msg ):
        self.write( self.encode_uint32(len(msg)) )
        self.write( msg )

        self.flush()
```

Listing B.107: simulator/python/simulator.py

```
#!/usr/bin/env python

#--- [Distribution] ----------------------------------------------------
# This file is part of the Disease Control System DiCon.
#
# Copyright (C) 2009  Sebastian Goll, University of Texas at Austin
# Designed and developed with the guidance of Nedialko B. Dimitrov
# and Lauren Ancel Meyers at the University of Texas at Austin.
#
# DiCon is free software: you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# DiCon is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with DiCon.  If not, see <http://www.gnu.org/licenses/>.
#-----------------------------------------------------------------------

import sys
import signal
import pickle
import traceback
import simulator_pb2
from StringIO import StringIO
from communicator import Communicator

```

```
32
33  class HangUp( Exception ):
34      pass
35
36
37  def hup_handler( signum, frame ):
38      raise HangUp
39
40
41  class Simulator( Communicator ):
42      def __init__( self ):
43          Communicator.__init__( self, sys.stdin, sys.stdout )
44          signal.signal( signal.SIGHUP, hup_handler )
45
46
47      def read_question( self ):
48          message = self.read_message()
49
50          question = simulator_pb2.Question()
51          question.ParseFromString( message )
52
53          return question
54
55
56      def new_answer( self ):
57          answer = simulator_pb2.Answer()
58          answer.failed = False
59          return answer
60
61
62      def write_answer( self, answer ):
63          self.write_message( answer.SerializeToString() )
64
65
66      def pickle( self, obj ):
67          buffer = StringIO()
68
69          try:
70              pickle.dump( obj, buffer, pickle.HIGHEST_PROTOCOL )
71              return buffer.getvalue()
72          finally:
73              buffer.close()
74
75
76      def unpickle( self, str ):
77          buffer = StringIO( str )
78
79          try:
80              return pickle.load( buffer )
81          finally:
82              buffer.close()
83
84
85      def process( self ):
86          answer = self.new_answer()
87          question = self.read_question()
88
89          try:
90              if question.type == simulator_pb2.SIMULATE:
91                  policy = []
92                  for node in question.q_simulate.node:
93                      policy.append( self.unpickle(node) )
94
95                  answer.a_simulate.reward = self.simulate( policy )
96
97              elif question.type == simulator_pb2.CHILDREN:
98                  path = []
99                  for node in question.q_children.node:
100                     path.append( self.unpickle(node) )
101
102                 for child in self.children( path ):
103                     answer.a_children.node.append( self.pickle(child) )
    )
104
105             elif question.type == simulator_pb2.DISPLAY:
106                 policy = []
107                 for node in question.q_display.node:
108                     policy.append( self.unpickle(node) )
109
110                 answer.a_display.display = self.display( policy )
111
112             else:
113                 raise IOError, "Got unknown message type."
114
115         except:
116             answer.failed = True
117             (type, value, tb) = sys.exc_info()
118             answer.a_failure.what = "".join( traceback.
    format_exception(type, value, tb) )
119
120         self.write_answer( answer )
121
122
123     def main( self ):
124         try:
125             while True: self.process()
126         except HangUp:
127             pass
128
129
130     def children( self, path ):
131         raise NotImplementedError
132
133
134     def simulate( self, policy ):
135         raise NotImplementedError
136
137
138     def display( self, policy ):
139         raise NotImplementedError
```

Appendix C

GNU General Public License

Version 3, 29 June 2007
Copyright © 2007 Free Software Foundation, Inc. http://fsf.org/
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights

or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

1. The work must carry prominent notices stating that you modified it, and giving a relevant date.

2. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

3. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

4. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined

with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

1. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

2. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

3. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

4. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the

place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

5. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized),

the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

1. Disclaiming warranty or limiting liability differently from the terms of sections

15 and 16 of this License; or

2. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

3. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

4. Limiting the use for publicity purposes of names of licensors or authors of the material; or

5. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

6. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent li-

cense under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the

covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies

that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM

TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

End of Terms and Conditions

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>

Copyright (C) <textyear>  <name of author>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.
```

```
You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program>  Copyright (C) <year>  <name of author>

This program comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.
```

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see `http://www.gnu.org/licenses/`.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read `http://www.gnu.org/philosophy/why-not-lgpl.html`.

Bibliography

[1] Merriam-Webster Online Dictionary (2009) on "epidemiology". `http://www.merriam-webster.com/dictionary/epidemiology`, December 2009.

[2] Cécile Viboud, Ottar N. Bjørnstad, David L. Smith, Lone Simonsen, Mark A. Miller, and Bryan T. Grenfell. Synchrony, waves, and spatial hierarchies in the spread of influenza. *Science*, 312:447–451, April 2006.

[3] Steven Riley and Neil M. Ferguson. Smallpox transmission and control: Spatial dynamics in Great Britain. *PNAS*, 103(33):12637–12642, August 2006.

[4] Vittoria Colizza, Alain Barrat, Marc Barthelemy, Alain-Jacques Valleron, and Alessandro Vespignani. Modeling the worldwide spread of pandemic influenza: Baseline case and containment interventions. *PLoS Medicine*, 4(1):95–110, January 2007.

[5] Steven Riley. Large-scale spatial-transmission models of infectious disease. *Science*, 316:1298–1301, June 2007.

[6] John S. Brownstein, Cecily J. Wolfe, and Kenneth D. Mandl. Empirical evidence for the effect of airline travel on inter-regional influenze spread in the United States. *PLoS Medicine*, 3(10):1826–1835, October 2006.

[7] Georgiy Bobashev, Robert J. Morris, and D. Michael Goedecke. Sampling for global epidemic models and the topology of an international airport network. *PLoS ONE*, 3(9):1–8, September 2008.

[8] M. Elizabeth Halloran, Ira M. Longini, Jr., Azhar Nizam, and Yang Yang. Containing bioterrorist smallpox. *Science*, 298:1428–1432, November 2002.

[9] Christophe Fraser, Steven Riley, Roy M. Anderson, and Neil M. Ferguson. Factors that make an infectious disease outbreak controllable. *PNAS*, 101(16):6146–6151, April 2004.

[10] Ira M. Longini, Jr., M. Elizabeth Halloran, Azhar Nizam, and Yang Yang. Containing pandemic influenza with antiviral agents. *American Journal of Epidemiology*, 159(7):623–633, April 2004.

[11] Ira M. Longini, Jr., Azhar Nizam, Shufu Xu, Kumnuan Ungchusak, Wanna Hanshaoworakul, Derek A. T. Cummings, and M. Elizabeth Halloran. Containing pandemic influenza at the source. *Science*, 309:1083–1087, August 2005.

[12] Neil M. Ferguson, Derek A. T. Cummings, Simon Cauchemez, Christophe Fraser, Steven Riley, Aronrag Meeyai, Sopon Iamsirithaworn, and Donald S. Burke. Strategies for containing an emerging influenza pandemic in Southeast Asia. *Nature*, 437:209–214, September 2005.

[13] Timothy C. Germann, Kai Kadau, Ira M. Longini, Jr., and Catherine A. Macken. Mitigation strategies for pandemic influenza in the United States. *PNAS*, 103 (15):5935–5940, April 2006.

[14] Neil M. Ferguson, Derek A. T. Cummings, Christophe Fraser, James C. Cajka, Philip C. Cooley, and Donald S. Burke. Strategies for mitigating an influenza pandemic. *Nature*, 442:448–452, July 2006.

[15] JTC1/SC22/WG21—The C++ Standards Committee. `http://www.open-std.org/jtc1/sc22/wg21/`, December 2009.

[16] Boost C++ Libraries. `http://www.boost.org/`, December 2009.

[17] Google Protocol Buffers. `http://code.google.com/p/protobuf/`, December 2009.

[18] GNU Multiple Precision Arithmetic Library. `http://gmplib.org/`, December 2009.

[19] Message Passing Interface Forum. `http://www.mpi-forum.org/`, December 2009.

[20] Google Protocol Buffers: Third-Party Add-ons for Protocol Buffers. `http://code.google.com/p/protobuf/wiki/ThirdPartyAddOns`, December 2009.

[21] Sukumar Ghosh. *Distributed Systems: An Algorithmic Approach*. Taylor & Francis Group, 2007.

[22] Gregory R. Andrews. *Foundations of Multithreaded, Parallel, and Distributed Programming*. Addison–Wesley, 2000.

[23] Robert M. Metcalfe and David R. Boggs. Ethernet: Distributed packet switching for local computer networks. *Communications of the ACM*, 19(7):395–404, July 1976.

[24] Jos C. M. Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335(2-3):131–146, May 2005.

[25] Hans Meuer, Jack Dongarra, Erich Strohmaier, and Horst Simon. TOP500 Supercomputing Sites. `http://www.top500.org/`, December 2009.

[26] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. *MPI—The Complete Reference: Volume 1, The MPI Core*. The MIT Press, 2nd edition, 1998.

[27] The University of Texas at Austin. Texas Advanced Computing Center: HPC Resources. `http://www.tacc.utexas.edu/resources/hpc/`, December 2009.

[28] Bjarne Stroustrup. *The Design and Evolution of C++*. Addison–Wesley, April 1996.

[29] Python programming language. `http://www.python.org/`, December 2009.

[30] ISO/IEC 14977:1996 (E). *Information technology—Syntactic metalanguage—Extended BNF*. International Organization for Standardization, Geneva, Switzerland, December 1996.

[31] Nedialko Dimitrov, Sebastian Goll, Lauren Ancel Meyers, Babak Pourbohloul, and Nathaniel Hupert. Optimizing tactics for use of the U.S. Antiviral Strategic National Stockpile for Pandemic (H1N1) Influenza. *PLoS Currents Influenza*, November 2009.

[32] Babak Pourbohloul, Armando Ahued, Bahman Davoudi, Rafael Meza, Lauren A. Meyers, Danuta M. Skowronski, Ignacio Villaseñor, Fernando Galván; Patricia Cravioto, David J. D. Earn, Jonathan Dushoff, David Fisman, W. John Edmunds, Nathaniel Hupert, Samuel V. Scarpino, Jesús Trujillo, Miguel Lutzow, Jorge Morales, Ada Contreras, Carolina Chávez, David M. Patrick, and Robert C. Brunham. Initial human transmission dynamics of the pandemic (H1N1) 2009 virus in North America. *Influenza and Other Respiratory Viruses*, 3(5):215–222, 2009.

[33] Christophe Fraser, Christl A. Donnelly, Simon Cauchemez, William P. Hanage, Maria D. Van Kerkhove, T. Deirdre Hollingsworth, Jamie Griffin, Rebecca F. Baggaley, Helen E. Jenkins, Emily J. Lyons, Thibaut Jombart, Wes R. Hinsley, Nicholas C. Grassly, Francois Balloux, Azra C. Ghani, Neil M. Ferguson, Andrew Rambaut, Oliver G. Pybus, Hugo Lopez-Gatell, Celia M. Alpuche-Aranda, Ietza Bojorquez Chapela, Ethel Palacios Zavala, Dulce Ma. Espejo Guevara, Francesco Checchi, Erika Garcia, Stephane Hugonnet, Cathy Roth, and The WHO Rapid Pandemic Assessment Collaboration. Pandemic potential of a strain of influenza A (H1N1): Early findings. *Science*, 324(5934):1557–1561, 2009.

[34] U. S. Centers for Disease Control. H1N1 flu clinical and public health guidance: Clinician guidance: Identifying and caring for patients. `http://www.cdc.gov/h1n1flu/identifyingpatients.htm#incubationperiod`, May 2009.

[35] Vernon J. Lee and Mark I. Chen. Effectiveness of neuraminidase inhibitors for preventing staff absenteeism during pandemic influenza. *Emerging Infectious Diseases*, 13(3):449–457, March 2007.

[36] James M. McCaw and Jodie McVernon. Prophylaxis or treatment? Optimal use of an antiviral stockpile during an influenza pandemic. *Mathematical Biosciences*, 209(2):336–360, October 2007.

[37] Vernon J. Lee, Kai Hong Phua, Mark I. Chen, Angela Chow, Stefan Ma, Kee Tai Goh, and Yee Sin Leo. Economics of neuraminidase inhibitor stockpiling for pandemic influenza, Singapore. *Emerging Infectious Diseases*, 12(1):95–102, January 2006.

[38] Aoife Doyle, Isabelle Bonmarin, Daniel Lévy-Bruhl, Yann Le Strat, and Jean-Claude Desenclos. Influenza pandemic preparedness in France: modelling the impact of interventions. *Epidemiology and Community Health*, 60(5):399–404, January 2006.

[39] Belinda Barnes, Kathryn Glass, and Niels G. Becker. The role of health care workers and antiviral drugs in the control of pandemic influenza. *Mathematical Biosciences*, 209(2):403–416, October 2007.

[40] Susy Hota and Allison McGeer. Antivirals and the control of influenza outbreaks. *Clinical Infectious Diseases*, 45(10):1362–1368, October 2007.

[41] Marc Lipsitch, Ted Cohen, Megan Murray, and Bruce R. Levin. Antiviral resistance and the control of pandemic influenza. *PLoS Medicine*, 4(1):111–121, January 2007.

[42] Free Software Foundation, Inc. GNU General Public License. `http://www.gnu.org/licenses/gpl.html`, December 2009.

[43] Free software foundation. `http://www.fsf.org/`, December 2009.

Vita

Sebastian Goll was born in Würzburg, Germany. After completing his *abitur* at Städt. Mozart- und Schönborn-Gymnasium Würzburg, Germany, he entered Julius-Maximilians-Universität Würzburg, Germany, in October, 2004. He finished his *vordiplom* studies in Computer Science and Physics in September, 2007. In August, 2008, he entered the Graduate School at the University of Texas at Austin.

Permanent Address: Estenfelder Str. 65
97078 Würzburg
Germany

E-mail Address: goll@physics.utexas.edu

This thesis was typed by the author in Emacs and typeset in LaTeX 2_ε.