The Dissertation Committee for Dylan Conrad Pfeifer Certifies that this is the approved version of the following dissertation:

Parallel and Distributed Cyber-Physical System Simulation

Committee:

Jonathan Valvano, Co-Supervisor

Andreas Gerstlauer, Co-Supervisor

Derek Chiou

Ranjit Gharpurey

Gian Gerosa

Parallel and Distributed Cyber-Physical System Simulation

by

Dylan Conrad Pfeifer, B.A.; B.S. Math; M.S.E.

Dissertation

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

Doctor of Philosophy

The University of Texas at Austin

December 2013

Dedication

To my parents, Walt and Jeri Pfeifer.

Acknowledgements

This work concludes a journey that has required the faith and support of many participants. Acknowledgement first goes to my research supervisor, Jonathan Valvano, who took me on as a Ph.D. student and proposed a course of research in the hybrid simulation of hardware and software systems, which evolved into the rich study of parallel and distributed simulation. Next, my co-supervisor, Andreas Gerstlauer, and committee members Derek Chiou, Gian Gerosa, and Ranjit Gharpurey deepened the work with supportive and consistent challenges during the course of research and discovery. Jonathan and Andreas provided substantial direction as well guiding the conference and journal publications resulting from the work.

While attending conferences and presenting papers during the research, I had great opportunities to meet recognized researchers in the field of parallel and distributed simulation, particularly Edward Lee of U.C. Berkeley, Richard Fujimoto of Georgia Tech, and David Jefferson of Lawrence Livermore National Laboratory. I also had the fortune to meet Christoph Grimm of SystemC-AMS development, Rishiyur Nikhil of Bluespec, Inc., who encouraged me to give a faculty talk at the University of Manchester, and Jim Garside of the University of Manchester, who was kind enough to facilitate the talk. These were great experiences, and these research leaders were kind to me and attentive to my ideas and questions.

Grateful acknowledgement is due to Intel Corporation, which allowed me to pursue Ph.D. research while also working as an Intel employee through the years 2009-2013. My managers, Sanjoy Mondal, Pankaj Kukkal, Nick Samra, and Haytham Samarchi were supportive during critical times of the work and committed to seeing me reach the final phases of the degree.

Family and friends have provided unflagging support, particularly my parents, brother, and grandmother, as they have in everything I have endeavored. The doctoral tradition established by my grandfather and continued through my mother and aunt helped me feel I could complete the journey, and my grandmother provided superior wisdom from many years of faculty roles. My father helped grow my imagination skills at a young age with discussions about the sciences, and my paternal grandmother started teaching me very early. The indefatigable support of my mother and father through the journey to doctorate can only be compared to the divine.

In that direction, I must acknowledge a faith in and gratitude towards a higher power, which created this universe with sufficient complexity that Ph.D. degrees may continue to be earned by students for some time to come. The acknowledgement of this creator is the beginning of knowledge. We are granted a partnership, such that if we try, we are met with wonders.

Parallel and Distributed Cyber-Physical System Simulation

Dylan Conrad Pfeifer, Ph.D.

The University of Texas at Austin, 2013

Supervisors: Jonathan Valvano and Andreas Gerstlauer

The traditions of real-time and embedded system engineering have evolved into a new field of cyber-physical systems (CPSs). The increase in complexity of CPS components and the multi-domain engineering composition of CPSs challenge the current best practices in design and simulation. To address the challenges of CPS simulation, this work introduces a simulator coordination method drawing from strengths of the field of parallel and distributed simulation (PADS), yet offering benefits aimed towards the challenges of coordinating CPS engineering design simulators. The method offers the novel concept of Interpolated Event data types applied to Kahn Process Networks in order to provide simulator coordination. This can enable conservative and optimistic coordination of multiple heterogeneous and homogeneous simulators, but provide important benefits for CPS simulation, such as the opportunity to reduce functional requirements for simulator interfacing compared to existing solutions. The method is analyzed in theoretical properties and instantiated in software tools SimConnect and SimTalk. Finally, an experimental study applies the method and tools to accelerate Spice circuit simulation with tradeoffs in speed versus accuracy, and demonstrates the coordination of three heterogeneous simulators for a CPS simulation with increasing model refinement and realism.

Table of Contents

List of Tables

List of Figures

List of Illustrations

CHAPTER ONE. INTRODUCTION

Cyber-physical systems (CPSs), defined as systems that integrate computation and physical processes, are becoming increasingly important for their transformative potential. Inheriting the field of embedded systems, but offering more distribution, communication, and computation capabilities, CPSs suggest new engineering and scientific opportunities as the number computational elements per device grows while devices shrink in terms of power requirements, cost, and size. The transformative power of what CPSs may have to offer in terms of controlled, coupled computation and physical processes has the "potential to dwarf the 20th century IT revolution" by virtue of ubiquity and impact [1]. The impact reaches everything from medicine to civil engineering, energy, defense, transportation, and smart consumer homes and devices [2].

Heterogeneous by definition, cyber-physical systems challenge their constituent disciplines, including electrical and computer engineering, computer science, mechanical engineering, biomedical engineering, and the traditional sciences. By consequence of their multi-domain composition, CPSs also challenge the discipline of simulation. Simulation is useful for systems where constructing a physical prototype and verifying functionality through build and test iterations is costly or perhaps impossible. Even if physical prototyping is manageable, simulation may still benefit the engineering design cycle, particularly for the computational side of the system during phases of integrated circuit design and verification, software development, and board-level circuit design. CPSs involving many independent computational elements interacting with the physical environment through transducers and actuators may have emergent properties that may not be discovered until the system is constructed and tested with real world physical processes. It is desirable to discover some of these properties in simulation rather than

reality, so that the software for the CPS is in an advanced state of functionality by the time the system is physically constructed. Therefore, simulation is beneficial not only to CPS behavioral discovery, but also to the best practices of CPS engineering. Robust, diverse, accurate, observable, and time-feasible simulation can enable adept and elegant CPS design.

1.1 Challenges of Cyber-Physical System Simulation

CPSs are composed of heterogeneous computation and physical processes [1]. At a system level, CPS simulation can require coordinating models of electrical hardware components (such digital processors, analog electronics, and mixed-signal application-specific integrated circuits), software components (real-time operating systems, software-based digital filters, software-based control, networking protocols), and physical models (such as transducers, dynamical systems, mechanical devices, and biological systems), each at potentially different levels of abstraction. These models may be simulated with continuous differential equation-based mathematical models of world physical processes, such as fluid dynamics or electromagnetics, and discrete based models for computational components. The challenge of mixing these components from different engineering and modeling domains for CPS simulation is called the "heterogeneous domain" challenge.

No single simulator or model of computation arguably spans the range of components that must be simulated by this challenge. For example, a cycle-estimating processor instruction set simulator may not be adequate for simulating an electrical network at the voltage and current level, but may be adequate for simulating the register and memory state of a processor for software development. A lumped element circuit simulator such as Spice [33], while popularly used for simulating circuits at the electrical level, may not be sufficient for simulating world-physical effects in CPSs that may need

to be modeled with space and time resolution, such as computational electromagnetics (CEM) or computational fluid dynamics (CFD). Finally, gate-level, clock-cycle accurate simulators supporting languages such as Verilog or VHDL can simulate large-scale VLSI circuits with gate-level resolution, but may not be sufficient for simulating high level software running over a virtual model for minutes of simulated software time.

Each of these domains, models, and simulation environments can be combined and coordinated to overcome some of their individual limitations. Mixes of simulators specializing in individual subsystems and modeling domains can increase the modeling range of simulation compared to a single simulator solution. Multiple simulator communication and coordination is both a benefit and challenge for system-level CPS simulation. The challenge arises as a consequence of model and model-of-computation diversity in CPSs, and as a consequence of simulation time cost. As the number of modeled components in a system increases, the time cost of simulation for some single simulators may increase exponentially. For some simulations, it may be possible to reduce the time cost of sequential simulation by partitioning the system among parallel, independent simulators, or leveraging algorithmic parallelism in the simulator where it exists. However, parallel simulation can introduce even more simulator communication and synchronization challenges.

Because software components (firmware, real-time operating systems, middle-ware, communication stacks, protocols, digital filters, and so on) are significant components of CPSs [4][5], simulation of software interacting with virtual models is another challenge to CPS simulation. Software is conventionally developed and modeled with a debugger running over a real, emulated, or virtual target such as a processor instruction set simulator.

Debugging software over virtual processor targets that include extra-processor models (like interfacing electronics, discrete hardware, transducers, and physical processes) is a challenge to system-level CPS simulation. Software debugging requires the ability to insert breakpoints into the debugger, single-step the program counter, or stop the debugger in time to inspect the target processor state, such as registers and memory. This is the source level debugging requirement of CPS software simulation. Yet, a CPS includes more components interacting with the software than just the simulated processor state. So, source-level debugging of software at system-level CPS simulation requires the ability to stop and inspect the state of other simulated models running outside of the processor target model that are affected by the simulated software.

Also, a CPS may include not just one processor target and software stack, but many virtual processors such as with a multi-core system, sensor grid, or multi-node industrial automation network. To support source level software debugging, extra-processor models must be able stop in time with each CPS software component being simulated and debugged, and then be able to resume without losing state or having state become altered by the pause and inspection. This is a distributed breakpoint problem across coordinated models, compared to single target breakpoints.

The software challenge of CPS simulation is further elaborated in [1] and [3]. Traditional embedded systems may couple computation with physical inputs through processor interrupts or cyclic polling. Testing software behavior to these inputs in simulation over a virtual processor model requires being able to pass extra-processor state into the processor model from a simulate peripheral in order to generate a simulated interrupt as it would occur in real hardware. For a CPS simulation, it is desired that these inputs actually come from simulated world physical process models acting through modeled transducers and peripherals. So, model coupling is a challenge because the

extra-processor models may be entirely different models of computation than the processor models. Therefore, just as CPSs challenge the scientific intersection of physical processes and computation interacting in real life, so also do they challenge the intersection of physical processes and computation modeled together in simulation.

1.2 Leading Coordination Solutions and Limits with CPS Simulation

Each of these challenges places CPS simulation firmly in the field of parallel and distributed simulation (PADS) [6][7][8], but with new challenges of diverse model coordination. Since the parallel and distributed simulation (PADS) challenge has been well represented in the modeling and simulation literature over the past three decades, with fundamental results by [6][7][8], it is important to identify contributions that may also benefit distributed CPS simulation.

PADS methods fall into categories of conservative or optimistic simulator coordination, or a mix of both [30]. These can be implemented with conservative messaging schemes such as the Chandy/Misra/Bryant-style null messages for conservative coordination [7], the numerous lookahead-based schemes for conservative coordination inspired by the Chandy/Misra/Bryant solution [36], or optimistic coordination approaches inspired from the Jefferson Time Warp solution [65].

Two important solutions among PADS implementations that inherit strengths of these approaches are the United States Department of Defense originated “High Level Architecture” (HLA), which has become IEEE standard 1516 [19], and the set-theoretic Discrete Event System Specification (DEVS) formalism, originating from Bernard Ziegler [20], resulting in tools such as Oak Ridge National Lab’s ADEVS suite [21].

1.2.1 THE IEEE 1516 HIGH LEVEL ARCHITECTURE (HLA)

HLA has shown particularly effective with large-scale, real-time, distributed military simulations with humans-in-the-loop (HIL) in the simulation federation. HLA's success in military simulation and training has earned it recognition as "the most influential standard in the field of distributed simulation" [22]. HLA is an architectural specification, and implementations offer simulator synchronization through the time management services of the Run-Time Infrastructure (RTI) specification of the architecture, which controls when time regulated federated simulators may advance in time [23]. Federates declare a Federation Object Model (FOM) of signals they will exchange and their attributes, and the HLA RTI supports techniques from the PADS literature for combined conservative and optimistic distributed simulation [23].

However, HLA has not yet been widely employed in engineering system design, embedded systems, or CPS system-level simulation to the definitive level of contribution it has provided for defense simulation and training. RTI software ambassadors for CPS components, for example, presently lack widespread instantiation among popular engineering software debuggers, logic simulation tools, and design automation tools. Yet, the potential for HLA as an embedded system and CPS simulation solution is recognized [27]. Also recognized is the outstanding effort to enable it by connecting tools such as VLSI electronic design automation tools into an HLA federation [24]. A shift of support for HLA RTI plugins among the major VLSI design vendors such as Cadence [25] and Synopsis [26] could signal a possible EDA industry migration to the solution. While Matlab/Simulink [21] now supports an HLA-Toolbox, and Matlab/Simulink is used as a numerical simulation federate in some reports [30], in CPS simulation we also desire to simulate numerous electronic components, such as ones widely supported by Spice models [31]. The authors in [27] state that their technique to

transform Simulink models into an HLA federate merits improvement, supporting only fixed time step advancement, for example.

Therefore, for complexity challenges of integrating numerical system simulation, software debuggers, and VLSI design automation integration into RTI-enforced coordination, HLA as stand-out solution to system-level design and simulation coordination for CPS engineering is open to be demonstrated. However, it is clearly valuable to CPS simulation for the multitude of world effects by HLA-compliant simulators offered and its IEEE standardization [19]. Its primary drawback to CPS simulation is that many diverse simulators must be coordinated to achieve a broad reach for CPS modeling and simulation, and the HLA RTI interfacing approach demands very tight integration with simulator runtime kernels. The tight simulator-level interfacing required to coordinate a closed architecture simulator with an HLA RTI may be an important market and research time cost. For CPS simulation, we volunteer a solution with coordination fidelity (attention to time synchronization and causality) and an interface specified primarily in the model definition layer rather than simulation kernel layer. We also seek to impose fewer and more simple functional interfacing requirements on federated simulators.

1.2.2 THE DISCRETE EVENT SYSTEM SPECIFICATION (DEVS)

The Discrete Event System Specification (DEVS), introduced in 1976 [20], focuses on model formalisms and formal algorithms to correctly simulate them, thereby separating as much as possible the art of modeling from the art of simulation. In this way a model can be verified to the degree it complies with a DEVS formalism, and a simulator can be verified to the degree executes a DEVS algorithm. The strength of DEVS is that it defines model interfacing channels for a variety of set theoretic models,

allowing for a general modeling specification with closure under model composition. When any DEVS conforming model is expressed in the DEVS notation, it can be coordinated with any other model in the DEVS system through DEVS model channels. ADEVS ("A Discrete Event System Simulator") [21] is a coordination solution in a set of open source C++ libraries that offers hybrid and distributed co-simulation for models conforming to the DEVS formalisms [20].

The primary limitation of DEVS for CPS PADS is that models must be expressed in a DEVS formalism to participate. For the range of models and devices we should like to include in a CPS simulation, such as the many circuit-level devices modeled in Spice simulators [31], we may not have market or research time to remodel the components of the CPS in DEVS. Rather, we should like to pick up a model as given, or where expertly simulated in the best environment, and interface that model and environment to other models and environments. We call this the "interfacing approach."

1.2.3 Other Significant Solutions

Other notable solutions include SimBus/Xyce [48], a parallel VHDL and parallel Spice solution from Sandia National Labs, backplane based solutions (examples of which are [12][13][14]), and 2- to 3-simulator special case coordinations, instances of which are numerous in the IEEE and ACM literature. However, these classes of solutions can require internal modification of the coordinated simulator kernels, which may not be allowed when interfacing proprietary, closed architecture simulators. While there are numerous examples in the space of special purpose 2- to 3-simulator coordination solutions, a solution is desired with the generality of a DEVS or HLA class solution.

Hybrid languages also offer hybrid modeling and simulation, the most flexible of which is arguably SystemC-AMS [47]. However, no single hybrid modeling

environment, such as SystemC-AMS, Verilog-AMS, or VHDL-AMS [34] is sufficient to cover the range of CPS simulation, by virtue of the number of models (sometimes proprietary) that a CPS system must simulate, and the potential localization of domain expertise in established simulators. Another honored solution is the Ptolemy system [35], from the University of California at Berkeley. The Ptolemy system has been recognized for real-time embedded system design [1][3][40].

We seek, however, an interfacing based approach to CPS PADS, rather than uniform modeling or uniform simulator approach, mainly because practitioners in CPS simulation may not be able to remodel all desired components in a uniform modeling or single simulation environment. It is also impracticable to assume domain expertise in every domain of the CPS simulation on the part of the modeler or simulation integrator. Therefore, we select the approach of collecting existing and upcoming tools from multiple engineering simulation domains that best model components and systems in their domain expertise, and interfacing them. We call the effort of synchronizing and providing communication between different simulators the effort of "simulator coordination" for this work. The method we introduce compared to the DEVS or HLA RTI based coordination approach is a *dataflow based* approach. We apply the properties of a well-defined dataflow formalism, the Kahn Process Network (KPN) [29], to ensure scheduling and synchronization properties of simulator coordination, and we innovate on the data tokens of the KPN with a type called Interpolated Events (IEs) to provide the required time and causal fidelity of a PADS solution. This approach may lessen functional burdens where simulators are interfaced compared to existing solutions, while still providing model and simulator-independent generality.

1.3 Thesis Statement: The KPN-IE Method for PADS

The Kahn Process Network (KPN) and Interpolated Event (IE) method of parallel and distributed simulation (PADS) offers conservative and optimistic coordination for multiple concurrent, distributed, and heterogeneous closed architecture simulators and can reduce time managing functional requirements for the coordination backplane and connected simulators compared to existing PADS solutions. A protocol based on KPN and IE facilitating PADS may reduce the time advancement and application layer messaging traffic to the coordination backplane. Additionally, an implementation based on KPN and IE based protocols may reduce the functional interfacing requirements and optimistic support requirements for federated simulators compared to the existing solutions.

1.4 Original Contributions

In this thesis, original contributions are offered to the field of parallel and distributed simulation from the KPN-IE method. The contributions are described in the following categories.

1.4.1 Original Contributions to the Field of PADS

- The Interpolated Event (IE) data type, which when forwarded according to the rules of Kahn Process Networks (KPNs), enables PADS with adherence to the local causality constraint, without logical process scheduling requirements
- The Interpolated Event Input Port and Interpolated Event Output Port plugin specification, which offers a lightweight interface for models and simulators to be coordinated, and enables a PADS model-level interface for closed architecture simulators

- An optimistic time management algorithm using the KPN-IE formalism that can reduce simulator interfacing and messaging requirements compared to existing solutions

1.4.2 ORIGINAL CONTRIBUTIONS THROUGH ENABLING TECHNOLOGIES

- A simulation message protocol, "SimTalk," realizing the method of IEs, which captures, independently, in a dataflow network, the synchronization information necessary to coordinate the simulation, conservatively or optimistically
- A simulation backplane, "SimConnect," realizing the KPN formalism, which, when paired with the SimTalk protocol, may not incur the functional management requirements of existing interfacing-based solutions, but can still enable conservative and optimistic coordination of connected simulators
- A method using SimConnect and SimTalk to coordinate multiple Spice simulators without internal modification of the Spice kernel, with tradeoffs in simulation speed versus accuracy

1.4.3 SIMCONNECT AND SIMTALK FEATURES

- A lightweight solution for PADS coordination targeting the needs of cyber-physical system simulation
- A means to conduct conservative simulation without null message traffic overhead
- A means to offer optimistic simulation but with no additional functional requirements for simulators other than the ability to save a 1-deep history of simulator state. Anti-message queues and other Time Warp style overheads [65] are not imposed on the simulators.

- Reduction in backplane functional requirements for coordination, because SimConnect (SC) is not required compute the global Lower Bound Time Stamp (LBTS) for conservative synchronization or the Global Virtual Time (GVT) for optimistic simulation. This information is captured in the IE data streams, and bounds on it can be monitored with tracking counters.
- A means for dynamic resolution in the distributed simulation that can be controlled by any participating simulator, observer, or the coordinating backplane
- Well-defined, expressible trade-off equations between the speed of simulation and accuracy in terms of IE primitives
- Mathematical analysis on IE signaling information through zero-order hold interpolated events

1.5 Overview of the KPN-IE Method for CPS PADS

1.5.1 The Interfacing Approach

CPS simulation can benefit from an interfacing approach instead of a unified modeling approach or uniform simulation environment. The interfacing approach allows CPS simulation to benefit from established and specialized models and simulators with closed architecture. This can reduce the modeling burden on CPS researchers, because they may not have to port strong existing models into a new language, model of computation, or simulation environment. Rather, models and environments are interfaced through the Interpolated Event Input and Output Port specifications introduced in Chapter Three. By taking the interfacing approach, the modeling capability of the KPN-IE method may increase on the cardinality of the power set of the set of all simulators for which Interpolated Event Input and Output Ports have been implemented.

1.5.2 THE DATAFLOW APPROACH

The KPN-IE method moves some of the coordination challenges of PADS into the dynamics of a KPN dataflow system, so that the simulation may be characterized by the interconnection dataflow. The KPN-IE approach is an observable, mathematically well-defined, distributable, and scalable data-flow network formalism that provides, through interpolated event data tokens, the synchronization and communication requirements of parallel and distributed CPS simulations. The capture, replay, and visibility of CPS system traffic in IE token format is convenient to a host of powerful software tools for analysis. The simulation can be completely characterized and evaluated in terms of events in the dataflow network and streams of interpolated events, rather than internal simulator events.

By design, the KPN-IE method achieves coordination and simulator advancement strictly through the KPN dataflow network and IE format, freeing the simulator coordination backplane from specific simulator object management and internal simulator time management. This can simplify the effort of implementing coordination software for a simulator.

1.5.3 SIMPLIFIED INTERFACING

A challenge of HLA RTI interfacing is that the RTI interface can be tightly coupled to the simulator internal kernel and time advancement. Simulator time advancement is managed explicitly by the external controlling software agent, the RTI. Because a simulator software layer may be proprietary, or of sufficient complexity that interfacing to an HLA RTI implementation through RTI and Federate Ambassadors (software interfaces) exceeds the budget of a researcher or the simulation vendor, the Interpolated Event Input and Output Port specification attempts to reduce the burden of simulator interfacing. By requiring only a user-level, device-model software interface

from the simulator and the ability to include OS-level libraries and compile OS-level system calls for tasking and network communication primitives, the IE port specification can be adept from a software engineering perspective. Most simulators offer a user-level, device-model software interface as a service, allowing users to monitor simulator signals and time, assign signals in time, and schedule events, but without exposing internal proprietary simulator time management software or intellectual property.

1.5.4 SIMPLIFIED MESSAGING

The KPN-IE method combines synchronization and communication in the IE token format, potentially reducing the messaging burden for conservative and optimistic coordination. With the HLA RTI, in addition to implementing callback functionality for signal update messages, the simulator must implement the Time Advance Request function or its sibling functions [74], which are separate messaging calls to the HLA backplane from the object attribute update functions. For a simulator time advance cycle, many signal update callbacks may be issued in addition to the time advance messaging requests. In KPN-IE method, only the IE token format is used for signal and time information, which may reduce the backplane messaging traffic at the application level. This traffic does not include the messaging costs of carrier technology, such as TCP/IP, or MPI, or the link and physical layers to the backplane.

1.5.5 ANALYSIS

KPN-IE offers a means for mathematical analysis of communication streams, because IEs are formally zero-order hold (ZOH) interpolators for continuous signals. This interpolation, along with the IE port specification, enables closed-form analytical expressions for tradeoffs in simulation resolution versus speed, and simulation resolution

versus accuracy for classes of model conditions. These expressions are constructed in Chapter Three.

Another important benefit of the IE token and port specification is that bounds on important conservative and optimistic coordination variables, namely the conservative Lower Bound Time Stamp (LBTS) and optimistic Global Virtual Time (GVT) values, automatically fall out of maintaining the KPN dataflow dynamics in the KPN-IE coordinating backplane. Additional messaging or simulator blocking is not imposed on the system to calculate these values, nor imposed on the coordinating backplane in a complex algorithm. The KPN backplane, therefore, is primarily a token router, focused on managing KPN and IE dynamics.

1.5.6 NON-EXCLUSIVITY

Lastly, KPN-IE method does not exclude participation in an HLA, or other hybrid simulation suite such as SystemC-AMS, Verilog-AMS, Ptolemy, or DEVS, if interfacing connectors such as SimTalk plugins are written for simulation suites that offer OS-level software interfaces. The goal of the IE port specification and SimTalk protocol is to be sufficiently light weight to interface to different environments with a short learning curve required from the simulation integration engineer.

1.6 Organization of Thesis

The organization of this work is as follows. Chapter Two covers primary material and concepts prerequisite to a discussion of PADS dynamics achieved through KPN-IE. The theory of modeling and simulation within the needs of CPS simulation is presented with no prior assumption of knowledge upon the reader. Important concepts and definitions that have persisted in PADS theory, composing a language for researchers in the field, are presented in preparation for the original work of Chapters Three and Four.

Terms such as state transition systems, modeling and simulation, conservative and optimistic coordination, logical processes, and timed, discrete event models are covered.

Chapter Three introduces the theoretical contributions to PADS provided by the KPN-IE method. Interpolated Events are introduced and defined, including operations on Interpolated Events, and the Interpolated Event Input and Output Port specification for allowing models to interface at the model-definition layer. Next, a simulation time cost function is constructed in terms of IEs and IE ports, leading to expressions for simulation speed versus IE resolution. Expressions for simulation error in terms of IE resolution are also constructed.

In Chapter Four, Kahn Process Networks are introduced, with elaboration of their formal properties, and the dynamics of Kahn Process Networks with Interpolated Event data tokens are covered. An example of conservative coordination with the KPN-IE method is given step by step. Next, the ability of the KPN network to yield bounded tracking of the important Lower Bound Time Stamp (LBTS) and Global Virtual Time (GVT) values for optimistic and conservative simulator coordination is described. An optimistic coordination algorithm is offered, which can reduce the anti-message queue maintenance burden on IE port-conforming models compared to existing leading Time Warp-inspired solutions [36][65]. Finally, a combined conservative and optimistic coordination scheme is offered based on the ability of the KPN to track bounds on LBTS and GVT in the IE token streams in the KPN FIFOs.

In Chapter Five, the software implementation of the KPN-IE method through original tools SimConnect and SimTalk is presented with examples of synchronization, dynamic resolution management, and example simulation configurations. The architecture of the SimConnect KPN backplane is discussed, and the SimTalk KPN-IE messaging protocol is discussed.

In Chapter Six, results from application of the KPN-IE method through SimConnect and SimTalk tools are covered for homogenous (many identical simulator) and heterogeneous (many different simulator) systems. The application of the KPN-IE method for parallel Spice circuit simulation at the model expression level is presented, offering a means of Spice acceleration by the coordination of many independent Spice simulators. Next, application of the KPN-IE method to simulate a software-managed, microcontroller PID/PWM based DC motor controller is presented. Tradeoffs in simulation resolution versus speed and accuracy are explored for both systems in each experiment.

Finally, in Chapter Seven, primary conclusions of the thesis are re-summarized, and areas of future work are offered based on new opportunities the KPN-IE method may offer to CPS simulation. A bibliography of important references in the field of CPS PADS is presented including peer-reviewed conference and journal articles generated by this work.

CHAPTER TWO. THEORY OF MODELING AND SIMULATION

This work concerns the theory and practice of engineering simulation for cyber-physical systems (CPSs). Simulation can be defined as the practice of building one system to discover the properties of another system. Simulation endeavors to reveal properties of a system desired to be observed without actually having to build the final system, but rather by building a *similar* system with *similarly* observable properties. Recommended fundamentals of simulation theory and practice are [20] and [36].

Several needs motivate the simulation of cyber-physical systems. First, during the engineering design cycle, repeated physical prototypes of the cyber-physical system may be expensive to build. For example, a system may contain new computational system-on-chip designs, which are costly to manufacture for each new production mask. A system may contain complex mechanical systems with high manufacturing costs or scarce materials. A system may contain complex software, the cost of which, due to writing and testing, can postpone timely delivery of the system. In simulation, however, these prototyping costs can be reduced by building and testing a *similar* system rather than repeated iterations of the physical one.

In the extreme, building the physical side of the CPS target system may not be possible. For example, it may be desired to know the transient dynamics of an early warning CPS system, such as a hurricane or earthquake warning system, but some physical parts of those systems (like hurricanes and earthquakes) are not possible to build. We must rather wait for those systems to occur and study them, with potential hazards. A similar *model*, however, might be constructed of those systems to provide insight into their behavior without the hazard of their physical reality. This model may

be tested with a model of the computational side of the CPS in the safety of a simulated world.

Other important motivations for CPS simulation are discovery, testing, and validation. For discovery, a model of a physical process may be so complex that emergent properties are not easy to predict. It may be beneficial to discover unstable, emergent properties of a system in simulation before they happen in physical reality. Additionally, some mathematical models of physical processes have no analytical, closed-form solution, but can only be simulated with numerical differentiation and integration [77] to observe their trajectories, or observed in real life.

For testing and validation, it is desired to know whether the system will function to a measure of confidence. Design flaws are desired to be identified before the system is constructed, particularly if resources only allow the system to be constructed once, or only allow the system to be exercised once (such as with some space probes). If a simulation exposes design flaws, there is a chance to eliminate them before the final system is built. In some CPS systems, such as biomedical devices [43], design flaws may be fatally dangerous. Simulation enables early testing of the system to eliminate design flaws so they are not present in the final system.

In each of these categories of need for simulation, a *model* of the system is created. The goal of model is to describe the system components and their behavior. A model is exercised in another system called a simulator. Therefore, modeling and simulation are distinct, but cooperating endeavors. A model and simulator may share a common underlying model of computation, which enables the simulator to correctly instantiate the model dynamics. This relationship is called a homeomorphism between the model and simulator [20], provided through the model of computation.

Modeling and simulation always encounter tradeoffs in detail and accuracy. A model may or may not be able to describe all of the system dynamics, and a simulator may or may not be able to instantiate all elements in the model and all element transitions specified in the model. Because of the limitations of models and simulators, errors can be introduced, which can be described as differences between a modeled or simulated property of a system and its composition and behavior in physical reality. These errors may be within a tolerance or critically misleading. The presence of error in each stage of the definition of modeling and simulation will be tracked as we introduce terms and concepts. We begin with a discussion of models.

2.1 Modeling

2.1.1 State Transition Systems

Modeling concerns the representation of elements and dynamics of a system. A starting point for models that evolve in some manner is the ***state transition system*** [68]-[72]. A state transition system is a pair $(S, \rightarrow)$ where S is a set of states and $\rightarrow \subseteq S \times S$ is a binary relation over S of transitions. If $p, q \in S$, then $(p, q) \in \rightarrow$ is notated as $p \rightarrow q$ and indicates there is a transition from state p to state q. State transition systems are a bridge from the domain of mathematics, where they are subclasses of "abstract rewriting systems," [68] to the realm of engineering, where they are a superclass for finite automata, labeled transition systems, and discrete event systems [68].

A transition system can be defined as a collection or set of elements (things to observe), with each element having a set of properties, or "states," assigned to it. The cardinality of the set of states S of elements in the system can be infinite, even uncountable (isomorphic to the set of real numbers), but for this work it will be finite. Let the system model be a set of elements E, and for each element e, let there be a set S_e

of properties the element could obtain at any instance of observation. That is for each element $e \in E$, at any point of observation, there is a state $s \in S_e$ given to the element. S_e could be the set of integers, reals, complex numbers, the set of colors, letters in the Latin alphabet, or a simple binary set $\{0, 1\}$. Specification of S_e is open to the modeler.

The set of states for a state transition system (denoted S) is defined as the set of all states the elements in the system may obtain. That is, at any one point of observation, the system state is represented by the tuple $\{s_0, s_1, \ldots, s_n\}$, where $s_i \in S_i$ is the state of element i in the system, for all elements n in the system, for all set of states S_i associated with each element. If the system is dynamic or state changing, it may occupy different points, each represented a state tuple $\{s_i\}$, at any point of observation. The state transition model, or the relation $\rightarrow \subseteq S \times S$, specifies how the system evolves from one state tuple to the next, giving the system behavior.

2.1.2 TIMED MODELS

Subclasses of state transition system models and how they are simulated are numerous [71], but for this work a primary classifier of systems will be the system element of time. A *timed* state transition system contains a model of time as an element of the system. For this work, timed models are models for which the state transition function is primarily a function of time, but may also be a function of time and also other state variables. For this work, no transition in a timed model is specified without the element of time associated with the transition. *Untimed models,* conversely, are not required to associate a time with every state transition. Untimed modes are a rich and important field in modeling research [35], but because this work concerns cyber-physical systems, which are engineered systems that transition in time, we are interested in timed models. The simulation of timed models may be time driven or event driven.

2.1.3 TIME DRIVEN MODELS

Time driven models have transitions with time as a primary stimulus of each system state transition. The simulation of time driven models involves choosing a point in the model time, inputting that time to the system, and evaluating the state transition function as a result of the input. The simulation then advances time to the next time point and repeats the process. The system may produce outputs during this cycle. Time driven models can be simulated discretely in a digital computer by capturing the system state in variables, choosing a start value t for time, and then evaluating the state transition function STF(t) for some finite subset of the discretely modeled time. Because the evaluation of STF(t) is countable and finite in a digital system, the simulation of the model can incur discretization errors. Digital simulation only proceeds through a finite subset of the system states, and each system state has finitely discretized element state.

2.1.4 EVENT DRIVEN MODELS

Event driven models are static until "something happens," that is, an event occurs [36]. The model receives notice of the event, and the transition function evaluates the new state as a function of the event. Event driven models may be discretely or continuously defined, but, like time driven models, their simulation is discrete in digital simulators. A timed, event driven model (TEDM) is one for which time is an element of the system state, and evolution of the system is specified in terms of events, not in terms of the advancement of time. Each event is associated with a point in time, since the model is timed, but simulation of the model involves creating an event e, and then evaluating the state transition function STF(e) as a function of the received event e. Background for event driven models and timed models is excellent in [36].

We can construct a classifier of models based on their representation of time and how they are simulated, whether time driven or event driven. Illustration 1 offers a

quadrant based classifier, where each quadrant represents a grouping of models. We can populate this classifier with examples from engineering simulation. Register transfer level (RTL) simulations are discretely simulated, discrete time, event driven models and simulations, with examples from Verilog and VHDL-based simulators occupying points in the +X, +Y, +Z quadrant. Simulations on analog computers occupy points in the -X, -Y plane. While simulation by analog components such as operational amplifiers may, in fact, still occupy discrete states in space and time (a topic beyond this work), we say that they are continuously simulated (CS) because time is not discretized in the analog computer in the manner it must be in a digital computer. Spice-based [31] circuit simulations can occupy the +Y, -X, -Z quadrant. Their models are expressed in continuous time differential equations, but their simulation is discretely conducted by algebraic difference equations in a digital machine.

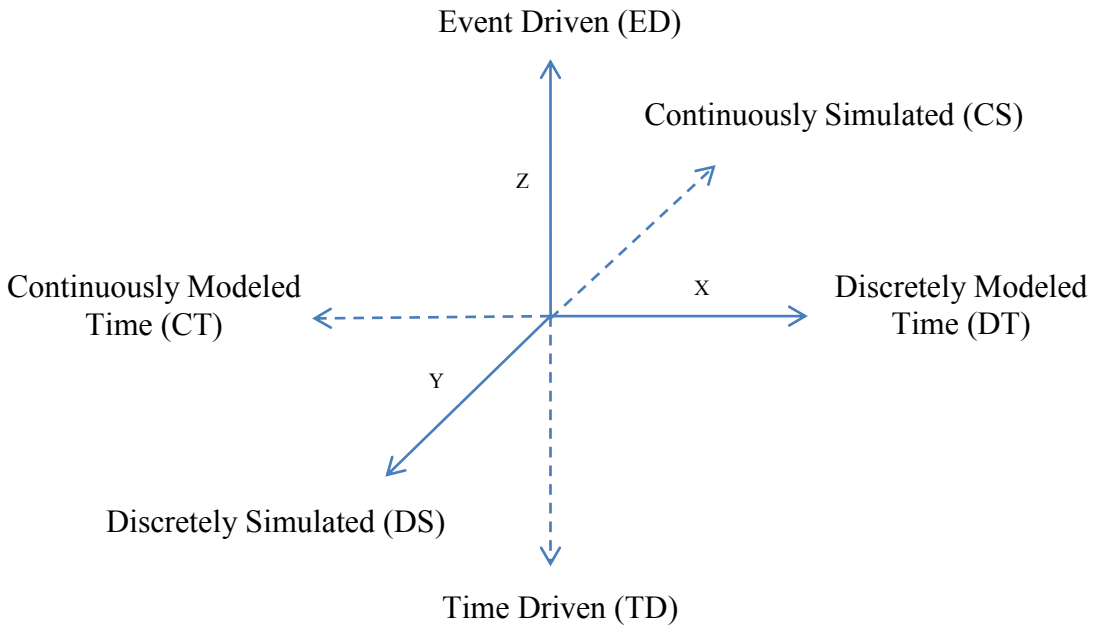


Illustration 1. Quadrant based categories of timed state transition systems

We are primarily concerned in cyber-physical systems with **timed, discrete time, discrete event, event driven** models. These are models conveniently simulated by a digital computer. As timed models, time is an element of the system state, but the advancement of the model is specified in terms of how the system reacts to events. A time change in the system without an event associated with it may be a modeled as a null event, so event driven models can capture a time driven simulation model of computation. The field of timed, event driven, discrete time modeling and simulation is rich [36], and we will continue to introduce terms, but we must account for categories of error early in the discussion.

2.1.5 ERRORS IN MODELING

The definition of modeling with state transition systems introduces errors as we depart from a physical reality. These are errors of representation. First, we may not know all the elements in the system, or the set of elements specified the model may be less than the set of all elements exercised in the simulator. This is the **error of simplification**. It can arise because we may not chose to study or model every element in the system, or because the simulator can only instantiate a subset of elements of the system model. Next is the **error of assignment**. This is the error that the set of states assigned to elements in the model to an element may or may not represent completely or exactly the set of physical states the element may obtain. The states of an element specified in the model may be a subset, or simplification, of the states it may obtain. For example, a switching element of a model may have a property state of {on, off}, or {0, 1}. However, its electrical signal level in the physical system might be a larger set (a subset of the real numbers assigned to model voltage, for example). So we have encountered an error of assignment by abstracting the element state.

Along with the state set of elements in the system, the state transition function may have representational error. The error of simplification implies that the state transition function may not be defined (or known) for all points in the system state. For example, we may say a state transition function is continuously defined if one of the elements of the system has a state set equivalent to R^1, the set of real numbers, and that the state transition function is surjective for that element. A state transition function may be discretely defined if all of the state sets for each element in the system are countable (equivalent to a subset of Z), or if the transition function is only defined for countable subsets of the system state. However, although a continuously defined transition function may be defined in closed form, when the model is exercised in a digital simulator, it becomes necessarily discrete. Therefore, we commit an error of simplification in the state transition simulation, because we cannot model all of the states specified in the system state set and transition function. It is sometimes necessary to simplify a model because the number of states and system elements increase simulation time costs when the model is simulated. It costs memory to store the model states, and it costs computation time to evaluate the state transition function. For some models these costs can increase nonlinearly as the number of elements in the system increases.

2.2 Simulation

The act of simulation is to realize the behavior of a model by instantiating elements of the model in simulator resources and executing the model transition function in a selected model of computation offered by the simulator. Executing the model of computation in the simulator takes a model through a state trajectory to observe simulated system properties. Simulation can be conducted with analog or digital computers, but this work concerns digital simulation. Separating the act of modeling and

the act of simulating allows separating the acts of verification and validation of the model and simulator [20]. Examining the model might ask if the right model is made, or if the errors of representation are tolerable. The art of simulating might ask if we are simulating the model correctly to its formalism within a tolerance of error [20][75].

The act of digital simulation is to realize a model and its transition function in a computationally realizable algorithm in a digital machine. The element properties are mapped to computer variable primitives and the transition function is mapped to instructions that the computer may execute on those variables. The act of simulation incurs the error of discretization. The error of discretization applies to both elements and the state transition function because the element property sets must be mapped to finite machine precision (encountering round-off error), and the state transition function must be mapped to finite machine arithmetic (encountering round-off and truncation errors) [76][77].

2.2.1 Accounting for Time in Digital Simulation

Two expressions of time are frequently given when accounting for time in a simulation. The time to execute the simulation, called wall clock time, or $\Delta \boldsymbol{T_{wall}}$, is the observed time elapsed for a simulation to start and stop as the observer experiences time. The simulation time, or $\Delta \boldsymbol{T_{sim}}$, is the time in the simulation that the model completes during a segment of wall clock time. In similar language, adopting the definitions of [36] physical time refers to time in the "real life" system being modeled. Simulation time is an "abstraction used by the simulation to model physical time" [36]. Simulation time is an element of the system state for timed models. Wall clock time refers to time during the execution of the simulated program, the time we experience as the simulation proceeds.

A timed, event driven model advances ΔT_{sim} by processing events. We can construct a real-time ratio, $\frac{\Delta T_{wall}}{\Delta T_{sim}}$, which quantifies how "quickly" a simulation advances. If $\Delta T_{wall} >> \Delta T_{sim}$, the simulation can be said to be "slow." If $\frac{\Delta T_{wall}}{\Delta T_{sim}} = 1$, the simulation is said to be running in real-time, or processing events and advancing as fast as the physical system would in "real life." If $\Delta T_{wall} << \Delta T_{sim}$, the simulation is said to run "as-fast-as-possible," meaning the system simulation advances much more quickly than the physical system would, processing events as quickly as it can. If $\Delta T_{wall} << \Delta T_{sim}$ and real-time simulation is desired, ΔT_{sim} can be scaled to match wall clock time by connecting the rate of event processing with a local real-time clock [36].

For cyber-physical system simulation and engineering simulation, it is most often the case that $\Delta T_{wall} >> \Delta T_{sim}$, due to complexity of state or compute time to evaluate the state transition function. Additionally, for CPS simulation in engineering product design, it is most often "as-fast-as-possible" simulation, rather than scaled real-time simulation, which proceeds at the same rate as wall clock time. Also, CPS simulation experiences a challenge in the difference in frequency of circuit level events that must be modeled versus software-level events, which might occur several orders of magnitude less frequently. Yet, both of these resolutions must be simulated with correct causal interaction with circuit-level and physical effects to reliably observe the system as it would behave if constructed.

2.2.2 Simulating Timed, Event Driven Models

Equivocating Time Driven and Event Driven Models

The simulation of time driven, discrete time models can be mapped to the simulation of event driven models if a change in simulation time is considered an event from the event driven viewpoint. In this mapping, the simulation chooses a countable,

finite subset of time (time points) to evaluate the state transition function, and then evaluates the system at those points. This is equivalent to making time an internal event queue, where the contents of the event queue are simulated time points when the simulation chooses to evaluate the state transition function STF(t). This mapping is important, because it can focus our view, without loss of generality, to event driven, discrete event simulation.

Discrete Time, Discrete Event Simulation

The timed, discrete event model (TDEM) is central to this work. Two prominent examples of TDEMs are "Logical Processes" [36] and the Discrete Event System Specification (DEVS) formalism [20]. We examine each in turn. First, we must define model inputs and outputs.

An input to a discrete event model is an event not generated internally by the model. An input is applied from sources "external" to the model. The input is applied through an input port, an abstract means to collect information into the model from sources not a part of the model set of elements.

An output is an event that can be generated as a function of the model's state and/or inputs. An output event may be fed back into the model through an input port for feedback. For these definitions, an output is considered to be an event that is sent "outside" of the model to another consumer.

A **timed event,** as described by [35], is a tuple (v, t), a value v and a tag t, taken from the product set V x T, where V is a set of values and T is a set of tags. Timed, event driven models advance in simulated time by processing timed events, whether generated internally in the model from its state transition function or supplied externally to the model through input ports.

Logical Processes

The Logical Processes viewpoint, covered by Richard Fujimoto in [36], points to a legacy of insights and techniques that influenced the IEEE 1516 High Level Architecture (HLA) standard [19] for distributed simulation coordination. Logical processes are event-driven processes with the following elements and definitions.

System state variables are elements "describing the state of the system" [36]. These are precisely the system elements and their properties defined in state transition systems. An **event list** contains events that are to occur at a "time in the simulated future" of the system [36]. An event list is a list of timed events ordered on *T*, the set of tags representing modeled time. A **local clock** denotes "the instant on the simulation time axis at which the simulation now resides" [36].

The simulation cycle of the sequential, discrete event Logical Process can be expressed as follows. While the simulation is in progress, remove the smallest time stamped event from the event list. Set the simulation clock to the time stamp of this event. Then execute all event handlers in the application to process the event [36].

The transition function consists of the actions of all the event handlers in the system model. Executing the event handlers moves the system from one point state to another and may add more events (internal events) to the event queue in the process. The state transition function for the Logical Process is therefore a function of *S* (the system state), *E* (the set of possible events), and *T* (the set of local time points).

The DEVS Formalism

The Discrete Event System (DEVS) formalism, attributed to Bernard Ziegler [20], is a set theoretic formal system. It attempts to classify and describe a model in terms of a DEVS set theoretic formalism, and then describe the concurrent simulation and composition of these models through formal interfaces. DEVS has a rich legacy dating to

1976 [20][21], and offers a means to separate the problems of verifying models and validating the correct simulation of models. The formalism has evolved in the PADS literature over the past three decades and is still actively researched [21][51][59][66].

Subtleties between LP and DEVS

The original atomic DEVS and Logical Process models encounter the challenge of simultaneous events, described as "events with the same time stamps" [36]. If a model receives two events of different values, but each with the same time stamp, which event shall the model process first? It turns out the correctness of the model can depend on the order in which the model processes the two events. Both system descriptions have been adapted in definition to handle simultaneous events.

Simultaneous events raise a challenge regarding event causality based on time stamps alone. A strongly causal system is said to have outputs that are strictly a function of present and past state and previous inputs. A weakly causal system has outputs that are a function of present and past state and present inputs. When simulating cycles of weakly causal systems, the simulation may depend on the order that events are received, even if they have the same time stamp. The receive order of simultaneous events may then not be repeatable over some networks from one run of the simulation to the next. Methods, such as extra lower precision bits added to event time stamps to differentiate between simultaneous events are offered in [36] and [51]. Another approach is to let the receiver decide between simultaneous events based on an internal priority [36]. A unified approach to DEVS and Logical Processes, offered by Nutaro [51], adds measures for simultaneity and causality in cycles of weakly causal processes.

2.2.3 DISTRIBUTED, PARALLEL, TIMED, DISCRETE EVENT MODELS

Distributed simulation means that the system simulation processes potentially run on spatially separate machines. Distributed simulation requires a means for processes to communicate events that they produce for each other. **Parallel** simulation means that the processes potentially execute concurrently. Parallel simulation requires a means to synchronize timed models so that processes receive and process timed events "when they are supposed to." That is, causality of events must be no different in parallel simulation compared to the causality of events in the same system simulated sequentially. Therefore, parallel and distributed simulation requires both communication and synchronization, terms this work combines together in the word "coordination."

Several needs motivate the creation and coordination of distributed and parallel TDEM process rather than sequential simulation of them over fast machines. First, the wall clock cost of simulating the state transition function STF(e) for event e may grow exponentially as the cardinality of S, the set of system states, increases for a model. This is an issue for models such electrical circuit models where the number of nodes or transistors in a system is large, as it is for VLSI systems. The reduction of STF(e) so that more elements of state may be simulated is a strong motivator for parallel and distributed simulation if STF(e) reduces for some domain over the set of parallel simulators.

Another motivation is model complexity. The system state S may be of such complexity that domain experts must define elements and properties, but cannot define all of them. For this reason, a model may be partitioned into domain groupings. The simulation of those models may only be available on proprietary simulators or only in a limited number or difficult to write from scratch simulation packages. Modelers may wish to focus on some elements of S and not burden themselves with other elements of S.

Another motivation is "level of abstraction." Level of abstraction can be a qualitative measure of the cardinality of the system state set and the domain and range of the system state function. A "low level of abstraction" may represent "many" elements of the physical system with refined state transition function, or a "high level of abstraction" may model "fewer" elements of the physical system at selected points of the system state (for accelerated simulation or more coverage of a subset of the system state). These terms are loosely applied, but good examples are conservative, electrical level modeling of a circuit for a low level of abstraction, and transaction-level-modeling (TLM) of the circuit for a high level of abstraction. It may be desired to model some components of the system state set at one level of abstraction and another set of system components at a different level. For example, it may be desired to model the processor in a system at an instruction-set-accurate level of abstraction, and the board circuitry of a system at an electrical signal level of abstraction. The error of simplification in representing the processor at a higher abstraction is not critical to the simulation purpose if we don't care about the internal circuit electrical levels of the processor, but only about the effect of the processor instruction events on the processor architectural state.

Therefore, the three motivating factors for distribution and parallelism are the reduction of the wall clock time cost to compute STF(e), heterogeneous model complexity, and heterogeneous levels of abstraction. Cyber-physical system simulation can encounter all three of these motivating factors for distributed and parallel simulation.

The Challenges of Distribution and Parallelism

Simulating a model or multiple models in parallel over distributed resources introduces challenges. These are the challenges of distribution, synchronization, and causality.

Because distributed simulation means that some aspects of the state transition function are evaluated over spatially separate compute resources, the models may need to exchange information (events) with each other. So, distribution brings the challenge of model and simulator communication.

Furthermore, because parallelism can mean that multiple logical processes execute in simulation with independent clocks local to each (and not modifiable or accessible by another process), parallel execution brings the synchronization challenge of control of model advancement in time. Note that distributed simulation does not necessarily mean parallel simulation. Parallel simulation means that the processes evaluate STF(*e*) and advance their local clocks potentially at the same wall clock time or over individual segments of simulated time. A simulation may still be sequential if distributed processes execute "one at a time," or if for any segment of wall clock time, for all segments of wall clock time, only one process is evaluating the state transition function and advancing its local time.

The primary hazard of the parallel simulation of timed, event driven models is event causality. With parallel simulation of timed models, each model has a *local copy* of time and simulation of the model advances independently. Therefore, at any instance in wall clock time, the local simulation time of each simulated model may be different. However, the parallel simulation must process events with the same outcome as if the simulation were sequential. If one model produces an event with a time stamp earlier than the simulation time of the model set to receive the event, then a *causality error* may occur. That is, a model has proceeded into simulated future beyond events that it could receive, so its state trajectory may no longer be correct.

Due to the challenge of causality, a means for *synchronizing* simulators is required, defined as a mechanism of ensuring causal correctness between processes.

Because models must communicate events to each other, the challenge of c*oordination* is the bipartite challenge of providing communication and synchronization service among parallel, distributed simulation processes.

Coupling Error

Parallel and distributed models are coupled by the coordination scheme assigned to the simulation. This introduces a possibility of coupling error, the third major source of error in modeling and simulation after representational error and discretization error. Coupling error is primarily a simulation-introduced error, where representational error is largely a modeling-introduced error. Coupling error may introduce in accuracies in the simulation of distributed models. Coupling error can introduce delay, which is both a wall clock time delay and simulation time delay. This delay can introduce causality and stability errors dependent on the coupling method if processes have independently running local clocks.

2.2.4 METHODS OF PARALLEL AND DISTRIBUTED, TIMED DISCRETE EVENT SIMULATION

The challenges of coordination in parallel and distributed simulation (PADS) and the assessment of error in the PADS have perpetuated the field as an active and open research area. The literature of PADS has grown its own lexicon in addressing open challenges, with signification contributions since the 1970s as concurrent computing resources became more accessible to researchers. Fundamental PADS concepts are next summarized, with recommended reading of Fujimoto [36].

Definitions

Local Causality Constraint: "A discrete-event simulation, consisting of logical processes (LPs) that interact exclusively by exchanging time stamped messages obeys the

local causality constraint if and only if each LP processes events in nondecreasing time" [36].

The local causality constraint (LCC) becomes challenging when a process does not know when it will receive the next external event. If a process consumes a future event *e* from an internal event queue, and advances its local clock to the time stamp of the event *e*, but later receives an external event with a time stamp *less* than event *e*, it violates the local causality constraint. For this reason, a process must determine which events in its event queues are "safe to process."

Safe to process: "An event *e* is 'safe to process' if the logical process can assure that it will not receive a future unprocessed event with a time stamp less than event *e*" [36].

An important result of the local causality constraint is that if each LP in a coordinated simulation obeys it, the "parallel/distributed execution will yield exactly the same results as a sequential execution of the same simulation program provided that events containing the same time stamp are processed in the same order in both the sequential and parallel execution" [36]. Some evaluation must be made for events with the *same* time stamp, because the order of execution of these events may affect the simulation outcome.

Lookahead: "If a logical process at simulation time T can only schedule new events with time stamp of *at least* T + L, then L is referred to as the lookahead for the logical process" [36].

Lookahead is an important concept in conservative simulation. A lookahead of *l* implies that there is a delay of at least *l* time units in a simulation for the process to output an event as a result of an input event. If a process *p* is at time *t* and has lookahead

l, other processes can know that they will not receive any events from process *p* until time $t + l$.

Zero-lookahead: A process has "zero-lookahead" if at logical time *t*, it may schedule *new events* at time stamp *t*.

Zero-lookahead implies that there may be no simulated time delay from consumption of an input event in a logical process to the production of an output event from the logical process caused by the input event. If a model has zero-lookahead, it may consume an input event at time *t* and produce an output event "caused" by that event also at time *t*. Zero-lookahead processes may lead to deadlock if connected in a dependency cycle. Also, zero-lookahead or small-lookahead processes degrade the performance of conservative simulations due to increased messaging per advancement of a segment of simulated time.

$LBTS_p$: The lower bound time stamp (LBTS) of future inputs for logical process *p* is a lower bound on the time stamp of any message that the process may receive in the future at the time of measure. The $\mathbf{LBTS_{min}}$ is the minimum $\mathbf{LBTS_p}$ for all processes *p* in the simulation at any point of observation.

Globally safe events: Events in the simulation at wall clock time *t* with time stamp less than or equal to $LBTS_{min}$ across all processes are safe to process.

If at any wall clock time *t*, each process in the simulation could know $LBTS_{min}$, it could execute all safe-to-process internal events (those less than $LBTS_{min}$) and meet the local causality constraint.

Conservative Simulation

A logical process is conservatively simulated if it only processes internal and received input events that are "safe to process." Conservative simulations process events

with time stamp *t* greater than or equal to local time if and only if it can be certain that the process will receive no new events with time stamp less than *t*. Conservative simulations require some means of communicating to processes which events are safe to process. Lookahead serves as means to evaluate event safety. A seminal conservative coordination scheme utilizing lookahead is the Chandy/Misra/Bryant null-message algorithm [7][36]. Processes send events with a null event value on all outputs every time the process local time advances so that consuming processes may have a lower bound (the null event time stamp) for when the process will next send an event. The algorithm can result in many null messages for processes with a small lookahead compared to the overall simulated time interval for the simulation.

Optimistic Simulation

A logical process is optimistically simulated if it may consume and process events speculatively and correct itself if a future event is received with time stamp less than the local clock. Optimistic simulations speculatively advance to processes future time events already scheduled, and if they receive an event with time stamp less than local time *t* (called a "straggler event"), there is a means, such as "rollback," to reverse the effects of the optimistic processing. The process must be able to cancel the effect of the optimistic event processing and return to an earlier state with time less than or equal to the time of the straggling event, *and* reverse effects of optimistic output events it has sent.

A seminal optimistic coordination scheme is the Jefferson Time-Warp scheme [65]. Jefferson Time Warp introduces the notion of "anti-messages" along with normal simulator event messages. A Time Warp Logical Process (TWLP) keeps internal queues with anti-message copies of all events it has sent out to other TWLPs through output queues. Anti-messages can be implemented with a sign bit. A normal message has a

positive sign bit and its anti-message, with exactly the same content, can have a negative sign bit. When a TWLP receives a straggling message and must rollback to an earlier time point, it must also undo the effects of all the speculative output messages it has sent. It does this by forwarding all anti-messages in its anti-message queues. When a TWLP receives an anti-message in an input queue, it "annihilates" its corresponding normal message (borrowed from the concept of particle/anti-particle annihilation). In this way a TWLP can cancel speculatively forwarded output events. There are other subtleties in this method, such as cascading rollback and the cancellation of already processed events, so [36] and [65] are recommended reading.

Optimistic simulation originating with Jefferson Time Warp adds the notion of Global Virtual Time (GVT). GVT is considered the earliest time in the simulation that a rollback may occur. Similar to $LBTS_{min}$, events in the simulation with time stamp less than or equal to GVT are safe to process and retire without fear of receiving a straggling event.

Global Virtual Time: "Global Virtual Time at wall clock time *t* (GVT_t) during the execution of a Time Warp simulation is defined as the minimum time stamp among all unprocessed and partially processed messages and anti-messages in the system at wall clock time *t*" [36].

Algorithms for determining GVT and the $LBTS_{min}$ of the entire system are critical to determining which events in the simulation are safe to process. Some coordination schemes require centralized blocking and additional simulator messaging to determine GVT and the $LBTS_{min}$ in the simulation [36].

The KPN-IE dataflow solution, however, may not impose upon coordinated simulators the same functional overhead required to support optimistic or conservative simulation. Because event messages are limited to interpolated event (IE) messages

coordinated among simulators with the rules of Kahn Process Network, calculation of bounds on $LBTS_{min}$ and GVT fall automatically out of implementation of the dataflow network, potentially reducing messaging burdens on the simulators and the coordination scheme. This tracking is explained in detail in Chapter Four.

2.3 Chapter Summary

This chapter provided an introduction to fundamental concepts in the field of modeling and simulation theory. Models for this work were derived from state transition systems, a formality sufficient to include subclasses of timed, time driven, event driven, discrete event or discrete time models. Event driven, discrete time, and discrete event models are central to this work. Fundamental concepts of timed, event driven models were given, and fundamental concepts of error encountered in modeling were introduced. Two important solutions for parallel and distributed simulation (PADS) were mentioned, the DEVS formalism [20] and the IEEE 1516 HLA standard [19]. Finally, fundamental definitions in the field of PADS were given. These definitions are important for understanding the dynamics of the KPN-IE method in Chapter Four.

CHAPTER THREE. INTERPOLATED EVENTS AND PORTS

This chapter introduces Interpolated Events (IEs) as a communication format and Interpolated Event ports as an input and output mechanism for parallel and distributed simulation (PADS) of cyber-physical systems (CPSs). Interpolated Events capture signal values and assign a segment of simulated time over which the signal is declared to be constant by the signal producer. Interpolated Events and their properties are formally defined, followed by the Interpolated Event Input and Output Port specification for sending and receiving IEs in a logical process model of computation. An equation for the time cost of simulation with IEs and IE ports is then constructed, followed by a discussion of categories of simulation error possible IEs and IE ports.

3.1 Interpolated Events

Interpolated Events are an important concept in this work. A conventional event inherited from the literature is a tuple (v, t), where v is from a set of values V, and t is from a set of tags T [35]. An ordering on conventional events can be associated with an ordering on T, particularly if T is R^1 under the Euclidean metric. An Interpolated Event (IE) is a 3-tuple, (v, t, t'), where v is from a set of values V, and t and t' are from a set of tags T [10].

3.1.1 Definition and Properties

Interpolated Events are 3-tuple elements (v, t_m, t_n) of the product set $V \times T \times T$, where V is a set of values, and T is a set of tags. This nomenclature borrows from the value/tag "(v, t)" definition of a conventional event [35]. For a given Interpolated Event (v, t_m, t_n), the value v is defined to be constant on the half open interval $[t_m, t_n)$ specified in the IE, such that the tag set T is ordered. T is conventionally the real number set R^1 in

timed, event driven simulations representing the simulated time when an event occurs. For an Interpolated Event (v, t_m, t_n), the range [t_m, t_n) assigns a "stable" time to the signal value v for producers and consumers.

If a simulator receives an Interpolated Event (v, t_m, t_n), it may assume the value v is constant on the tag range [t_m, t_n), and not need to sample the value again until expiration time t_n. So, an Interpolated Event encapsulates both communication (the signal value) and synchronization (the start and end time). Mapped to nodes in a Kahn Process Network, simulators consume IEs, run, and produce IEs until the expiration tag of the last consumed IE, at which point simulators sample their FIFOs again for new IEs. If their input FIFOs are empty, simulators block, enforcing the local causality constraint, because each simulator cannot advance in time beyond the expiration tags of IEs on its input FIFOs. A feature of the sampling captured in the duration of an IE ($t_n - t_m$), or "**ΔIE**," is that tradeoffs in simulation speed versus accuracy may be studied. An IE assigns a stable value to a signal for a duration, during which local time a consuming simulator can operate on it without re-querying the value. The speed versus accuracy tradeoff can be statically or dynamically adjusted, as explored in [11][17][18].

3.1.2 Operations on Interpolated Events

We next define groupings and operations on Interpolated Events. These definitions enable techniques of optimistic and conservative simulator coordination where signals are communicated as streams of Interpolated Events.

IE stream: An IE stream { *IE* } is a set of IEs, {(v_0, t_{m0}, t_{n0}) … (v_k, t_{mk}, t_{nk})}.

An IE stream is **connected** if and only if for every t_n value in { *IE* } less than the max t_n value in the stream, there is at least one other IE' in the stream, different from the IE containing t_n, for which t_n is contained in [t_m', t_n') of IE'. That is, there is at least one

other IE for which $t_{m'} \leq t_n \leq t_{n'}$ for an IE (v', $t_{m'}$, $t_{n'}$). An equivalent statement is that for any t in the interval [$t_{m\ min}$, $t_{n\ max}$] in the stream, there is at least one IE containing t. That is, there is an IE (v', $t_{m'}$, $t_{n'}$) for which $t_{m'} \leq t \leq t_{n'}$.

An IE stream is **connected and non-overlapping** if it is connected and for every t_n value in { *IE* } less than the max t_n value in the stream, there is at most *one* IE' in the stream for which $t_{m'} = t_n$ for the IE', (v', $t_{m'}$, $t_{n'}$), and there are *no other* IEs in the stream for which $t_{m''} < t_n < t_{n''}$ for an IE'' (v'', $t_{m''}$, $t_{n''}$). Connected and non-overlapping IE streams have non-duplicate t_m values among the t_m values, and non-duplicate t_n values among the t_n values, and all IEs have non-zero ΔIE.

An IE stream is **connected and overlapping** if it is connected and for any t_n value in the stream less than the max t_n in the stream, there is *at least one* other IE in the stream (v', $t_{m'}$, $t_{n'}$) for which $t_{m'} < t_n \leq t_{n'}$. An equivalent statement is that for at least one time t in the interval [$t_{m\ min}$, $t_{n\ max}$] in the stream, there are at least two IEs in the stream containing time t.

IE output ports can produce connected and overlapping IE streams in tracking mode (Chapter 3.2.1). Corrected IEs overlap with previously posted IEs. IE output ports in sampling mode produce connected and non-overlapping IE streams.

IE splitting: The split operation on an IE divides its ΔIE interval. Let the IE be given by (v, t_m, t_n), and $t_m < t_n$. Let t_s be contained in the interval [t_m, t_n). The split operation split(IE, t_s) is a mapping from [t_m, t_n) -> [t_m, t_n) x [t_m, t_n) as follows:
split(v, t_m, t_n, t_s) = (v, t_m, t_s), (v, t_s, t_n). The split divides the IE into two connected IEs with value v connected at time t_s contained in [t_m, t_n).

IE concatenation: IE concatenation combines IEs. For two IEs given by (v_1, t_m, t_n), (v_1, t_n, t_o), $t_m < t_n < t_o$, the operation concat((v_1, t_m, t_n), (v_1, t_n, t_o)) produces (v_1, t_m, t_o). Concatenation lengthens a ΔIE segment for connected IEs of the same v value.

The act of concatenating a connected and non-overlapping IE stream can produce a stream with the same event structure but with fewer IEs. The act of splitting a connected and non-overlapping IE stream can produce the same event structure but with more IEs.

IE sharpen: The IE sharpen operation resolves overlapping IEs in a stream of IEs into a stream of connected and non-overlapping IEs. For any two overlapping IEs, (v_1, t_{m1}, t_{n1}), (v_2, t_{m2}, t_{n2}), we have $t_{m1} < t_{tn2} \leq t_{n1}$. Which IE should have precedence? If $\Delta IE_0 < \Delta IE_1$, we choose ΔIE_0 to have precedence, because it is more refined (smaller ΔIE). If the two t_m are equal for the two IEs, we simply choose the IE with the smaller ΔIE and discard the other. If they have the same ΔIE, we choose the one appearing later in LIFO order, and discard the other. If neither of these conditions, we let t_m be the smaller of the t_m values of the two IEs. We construct a new IE with v equal to the value associated with the IE with the smaller t_m. The IE has value v, t_m, and the t_n as the greater t_m of the two. Then we create another IE with v equal to the v of the later appearing IE (larger t_m), and with t_n of the last of the two IEs in LIFO order. The sharpening is then continued over each remaining overlapping IE in the stream.

3.2 Interpolated Event Ports

3.2.1 Interpolated Event Output Ports

The Interpolated Event Output Port (IEOP) is a timed, discrete event driven model that has the logical process internal clock tied to the model using the port. We define an abstract state machine for generating interpolated events or consuming them on model input and output ports. The IEOP can be an independent logical process as long as it receives synchronous local clock updates from its parent model (the model that uses the

port). IEOPs produce an interpolated event stream and have an event driven logical process structure as described in the following.

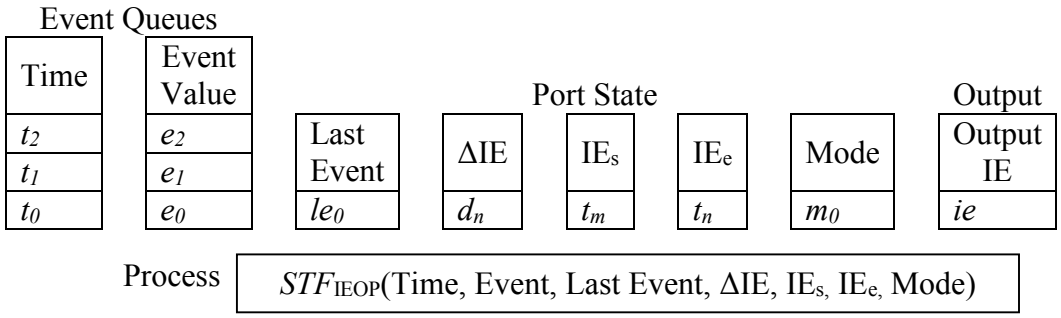


Illustration 2. State elements of the Interpolated Event Output Port

Illustration 2 shows the logical process, timed, discrete-event model for the Interpolated Event Output Port. The process contains two input event queues, named "Time" and "Event Value." It contains five state variables, "Last Event, ΔIE, IE_s, IE_e, and Mode." These can also be considered single item event queues rather than state variables for event-driven generality. The state transition function STF_{IEOP} is triggered when the Event Value queue receives an event. The function is blocked if any of the event queues are empty. The logical clock of the process advances with events in the Time queue. The state transition function determines an output when the time and event queues are non-empty. By design, because input queues are from a timed discrete event driven system, when an event is added to the event input queue, a corresponding time event with equal time stamp resides in the time queue representing the local time of the deposit of the event in the IEOP. Expressed in pseudo code, the transition function is:

```
if (time queue is non-empty) {
     if (mode = sampling) {
          if (pop(Time) ≥ tn) {
               if (Event queue is non-empty) {
```

```
                    Output IE v = pop(Event);
                    flush(Events);
               } else {
                    Output IE v = Last Event value;
               }
               Output IE t_m = t_m; Output IE t_n = t_n;
               t_m = t_n;
               t_n = t_n + ΔIE;
               Last Event value = Output IE v;
               flush(Events);
          } else {
               pop(Time);
          }
     }
     if (mode = tracking) {
          if (Event queue is non-empty) {
               Output IE v = pop(Event);
               // send correction on last output IE
               Output IE t_m = t_m;
               Output IE t_n = pop(Time);
               t_m = t_n;
               t_n = t_n + ΔIE;
          } else {
               pop(Time);
          }
     }
}
```

Illustration 3. Interpolated Event Output Port pseudo code example

Two output modes are available on the IEOP. In sampling mode, the process outputs an IE every ΔIE event samples. If no new events are queued each ΔIE interval, the last event forms the output IE. Sampling mode IEOPs can mask events if ΔIE is greater than the period of input events.

In tracking mode, the IEOP outputs a correcting IE if a new event arrives before ΔIE expires. Although output IEs will overlap in tracking mode, the IEOP will not miss any input events. Tracking mode can be used for correction of optimistic IE streams.

Two important aspects of the IEOP concern lookahead. If the LP using the port has non-zero lookahead K, and ΔIE = K, then the IEOP will behave as if in tracking mode and output an IE stream with no overlap. If ΔIE < K and divides K, then the IEOP will behave as if in tracking mode and output an IE stream with no overlap, but with more IEs than if ΔIE = K. If ΔIE < K and does not divide K, or if ΔIE > K and the port is in sampling mode, then the port may mask events. That is, the event will not output because it does not fall on the boundary of a ΔIE time interval.

ΔIE can be changed by the process using the port or by external IE consumers. The tuning ΔIE affects the speedup and accuracy of a parallel simulation and is an important configuration parameter. ΔIE provides an optimistic prediction of the stability of an event *e*. The ΔIE configuration declares the output will not change over the interval of ΔIE unless corrected in tracking mode.

3.2.2 INTERPOLATED EVENT INPUT PORTS

The Interpolated Event Input Port receives IEs from an external provider and creates internal events for the consuming logical process.

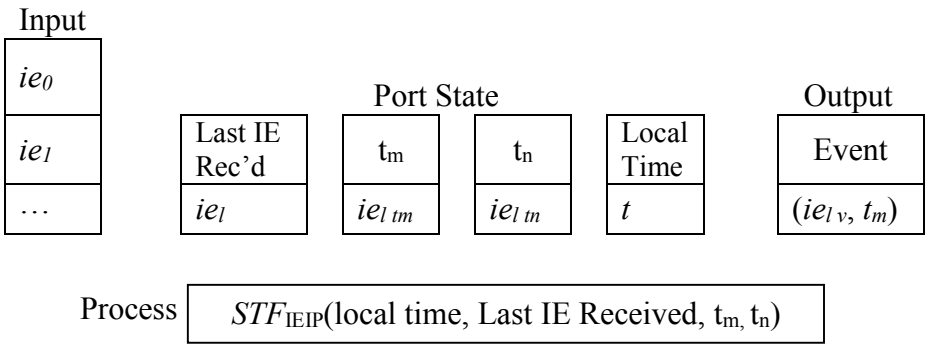


Illustration 4. Interpolated Event Input Port state elements

The Interpolated Event Input Port consumes IEs from an input source external to the model and produces conventional internal events for the TDEM logical process

connected to the port. The conventional event ($ie_{l\ v}$, t_m) is an event (v, t) where v is the value of ie_l and t_m is the ie_l t_m value. If the process local time is equal to the last t_n value of the last IE the port received, the port performs a ***process blocking read*** from its input source for a new IE. The blocking read prevents the attached TDEM from advancing local time beyond the time of incoming IEs, forcing the local causality constraint. If no new IEs are available, the process waits in a blocking executive. If a new IE is available, the port consumes the IE, updates the last IE received state, t_m and t_n states, and queues an internal conventional event in the TDEM model with value v equal to the received IE v value and the time of the event set to t_m. This event "persists" until expiration time t_n. If the model consumes the event at local time $t \geq t_m$, the port abstractly places another conventional event in the input queue with value (*v, local time*) so that the TDEM can never become blocked on an empty input event queue during duration $t_n - t_m$ for the last received IE. The TDEM process may safely execute any other internal events during this time and run independently because the IE value is declared unchanging. A pseudo code state transition function for the Interpolated Event Input Port is:

```
if (local time = tn) {
     while (dequeue(input IE) = empty) \
          TDEM process block;
     // a new IE' is available
     consume IE' (v', tm', tn') from input IE FIFO;
     Queue TDEM event (v', local time);
     // update states
     Last IE Received = IE'; tm = tm'; tn = tn';
} else {
     if (output queue = empty) {
          Queue TDEM event (Last Received IE v, local
          time);
     }
}
```

Illustration 5. Interpolated Event Input Port pseudo code example

Models with Multiple Ports

The KPN-IE method declares that ports for coordinated models may only be of these two types: IEIPs or IEOPs. This allows for analytic closed-form expressions of simulation speed versus IE resolution, for assuring the local causality constraint, and for other coordination benefits that will be introduced. Timed, discrete event models can have multiple Interpolated Event Input and Output Ports. The IE port models impose light weight requirements on the simulators, namely that they offer inspection of internal local time and internal events to be communicated through the port, that they offer an operating system (OS)-level system interface to implement the messaging scheme (format, verification, and carrier) and a process blocking OS primitive, and that they allow scheduling of future internal events. These interfaces may be provided in proprietary simulators to offer a means of user-based interfacing without exposing internal simulator source code or intellectual property.

3.3 A Simulation Time Cost Function with IEs and IE Ports

We can now formulate a detailed simulation time cost tradeoff function with IEs, IE ports, and timed, discrete event simulation. This time cost function will result in expressions for system simulation speed versus simulation resolution.

Total wall clock time cost of single process, timed, discrete event simulation

The timed, discrete event simulation time cost equation is the wall clock time required to simulate a sequence of events within a segment of simulated time. The expression is given in Equation 1.1.

$$\Delta T_{wall,\ \Delta T_{sim}} = \sum_{i=0}^{n} \Upsilon_{STF}(e_i, S) \mid e_i \in E_{\Delta T_{sim}} \quad \text{(Eq. 1.1)}$$

$STF(e_i, S)$ = the state transition function for event $e_i \in E_{\Delta T_{sim}}$ and system state tuple S

$\Upsilon_{STF}(e_i, S)$ = the wall clock time to compute $STF(e_i, S)$ for each e_i

$E_{\Delta T_{sim}}$ = the set of all events simulated by the model with time stamps contained in ΔT_{sim}

n = $\left|E_{\Delta T_{sim}}\right|$, the cardinality of $E_{\Delta T_{sim}}$

$\Delta T_{wall,\ \Delta T_{sim}}$ = the wall clock time cost for the simulation to advance ΔT_{sim} units of simulation time

Accounting for outputs in the simulation time of single processes

Equation 1.1 assumes that outputs are contained in the system state, meaning that an output is a copy or reference to an element of the system state and is known when $STF(e_i, S)$ is evaluated. However, an output may be a function of the system state, a map from *S* to a set of outputs, and evaluated whenever the model is to "output" an event. We use the DEVS notation for the system output function, λ. Because λ is a function, if system state S_1 equals system state S_2, then $\lambda(S_1) = \lambda(S_2)$. So, at most, λ is evaluated every $STF(e_i, S)$. So, accounting for the evaluation of outputs by the model, Equation 1.1 becomes:

$$\Delta T_{wall,\ \Delta T_{sim}} = \sum_{i=0}^{n} \Upsilon_{STF}(e_i, S) + \sum_{i=0}^{m} \Upsilon_{\lambda_i}(S) \mid e_i \in E_{\Delta T_{sim}} \quad \text{(Eq. 1.2)}$$

$\Upsilon_{\lambda_i}(S)$ = the time cost to compute the system output function

m = the number of times λ is evaluated in ΔT_{sim}

Partitioning E

We declare $|E|$, the cardinality of set E, which is the set of events processed by an event driven model, to be countable for the discrete event, event driven simulation. It may also be finite, but is not required to be (the simulation can always be stopped manually). $Y_{STF}(e_i, S)$, the wall clock time to evaluate $STF(e_i, S)$ for each event e_i in E, may dependent on the number of states in the system, the compute resources evaluating the event, and the event itself. In some simulations the cost may be known; in others it can only be measured or bounded.

We can partition the set E into categories of events. First, there are internal and external events. External events are events provided to the model, such as initial conditions or inputs, and are not generated by the model itself during simulation. Internal events are events created by the model for itself from the action of processing external events. So, $E = E_{ext} \cup E_{int}$, where $E_{ext} = \{e_{ext} \in E \,|e_{ext}\ is\ an\ external\ event\}$, $E_{int} = \{e_{int} \in E \,|e_{int}\ is\ an\ internal\ event\}$, and $E_{ext} \cap E_{int} = \emptyset$ by definition.

It is important to distinguish between pre-scheduled events and run-time originated events. Pre-scheduled events are events and times scheduled for a model in simulation before the simulation is run. These may be called "pre-scheduled time points." For example, if a model must evaluate state at minimum every Δt simulation seconds, then there will $\Delta T_{sim} / \Delta t$ are pre-scheduled time points for every segment of simulation time ΔT_{sim}. $STF(e_i, S)$ will be evaluated at each of the pre-scheduled points, where e_i will be the time point event.

We define inputs further to clarify the concept of external events. An input is a via into a model through which information is passed. For this work, inputs are restricted to arrive through IEIPs and contain IEs.

Input events are external events in that they are not originated by the model itself. Output events may be fed back into the input for feedback, but we identify this case separately. So, continuing to partition categories of external events, $E_{ext} = E_{inputs} \cup E_{presched}$, where $E_{inputs} = \{e_{inp} \in E \,|e_{inp}\ is\ an\ input\ event\}$, and $E_{presched} = \{e_{presched} \in E \,|e_{presched}\ is\ a\ prescheduled\ event\}$. $E_{inputs} \cap E_{presched} = \emptyset$ by definition. The set of inputs may be pre-specified, such as an event segment stimulus, but they arrive through input ports, where prescheduled events are pre-filled in the model event queues prior to the simulation. Therefore the set of input events and the set of prescheduled events are separate.

Expanded equation for abstract input and output ports and partitioned E

$$\Delta T_{wall,\ \Delta T_{sim}} = \sum_{i=0}^{L} \Upsilon_{STF}(e_i \in E_{input,\Delta T_{sim}}, S) + \sum_{i=0}^{M} \Upsilon_{STF}(e_i \in E_{internal,\Delta T_{sim}}, S) + \sum_{i=0}^{N} \Upsilon_{STF}(e_i \in E_{prescheduled,\Delta T_{sim}}, S) + \sum_{i=0}^{O} \Upsilon_{\lambda_i}(S) \quad \text{(Eq. 1.3)}$$

L = the number of external input events during ΔT_{sim}

M = the number of internal events generated during ΔT_{sim}

N = the number of prescheduled events during ΔT_{sim}

O = the number of times the output function is evaluated in ΔT_{sim}

Equation 1.3 accounts for the event category decomposition. Since inputs are restricted to arrive through interpolated event ports, they may be counted. Let ΔIE_i be the interpolated event duration for an interpolated event IE_i arriving in simulation time period ΔT_{sim}. Assuming ΔIE constant for interpolated events in ΔT_{sim}, there will be $\Delta T_{sim}/\Delta IE$ such interpolated events during that time period. Each interpolated event is an external

event for the model. So the interpolated event input ports add the wall clock time cost of servicing the port (receive costs) and the wall clock time cost of waiting for the next event, since inputs block at IE expiration times.

Accounting for Interpolated Event Input Ports

In Equation 1.4, the event category decomposition is accounted for with interpolated event inputs. *K* is generally one (the interpolated event is processed once as a single event with time stamp t_m from the *ie*), but the model may consume the event at multiple time points, re-evaluating the state transition function each time.

$$\Delta T_{wall,\ \Delta T_{sim}} = \sum_{j=0}^{J} \sum_{i=0}^{\Delta T_{sim}/\Delta IE_j} [\Upsilon_{IEIP}(ie_{i,j} \in IE_{j,\Delta T_{sim}}) + \Upsilon_{wait}(ie_{i+1,j}) + \sum_{l=0}^{K} \Upsilon_{STF}(e_l \in ie_{i,j}, S)] + \sum_{i=0}^{L} \Upsilon_{STF}(e_i \in E_{internal,\Delta T_{sim}}, S) + \sum_{i=0}^{M} \Upsilon_{STF}(e_i \in E_{prescheduled,\Delta T_{sim}}, S) + \sum_{i=0}^{N} \Upsilon_{\lambda_i}(S)$$

(Eq. 1.4)

J = the number of interpolated event input ports

K = the number of internal events created by $ie_{i,j}$

L = the number of internal events generated during ΔT_{sim}

M = the number of prescheduled events during ΔT_{sim}

N = the number of times the output function is evaluated

$\Upsilon_{wait}(ie_{i+1,j})$ = the wall clock wait time for the next input $ie_{i+1,j}$

$\Upsilon_{IEIP}(ie_{i,j} \in IE_{j,\Delta T_{sim}})$ = the wall clock time cost of servicing received $ie_{i,j}$

$IE_{j,\Delta T_{sim}}$ = the set of *ie's* received on port *j* during ΔT_{sim}

$\Upsilon_{STF}(e_l \in ie_{i,j}, S)$ = the wall clock cost of evaluating the state transition function for internal events created by received $ie_{i,j}$

A critical wait introduced in Equation 1.4 is $\Upsilon_{wait}(ie_{i+1,j})$, the wall clock time for the process to wait while it is blocked waiting for the next *ie* input on port *j*. Processes block for the local causality constraint and wait on new interpolated events. As such, $\Upsilon_{wait}(ie_{i+1,j})$ is the wall clock time required for the process sending $ie_{i+1,j}$ to advance $\Delta ie_{i,j}$ time points in simulation time plus the transmit delay of the IE information, which can be network dependent. We can see from Equation 1.4 that ΔIE and $\Upsilon_{STF}(e_i, S)$ are significant factors in the wall clock time cost of simulation where message transmit time and port processing time $\Upsilon_{IEIP}(ie_{i,j})$ are insignificant. Simulation resolution is changed by changing the interpolated event duration ΔIE on a port. We seek a relationship between ΔIE and $\Delta T_{wall,\ \Delta T_{sim}}$. Equation 1.5 accounts for interpolated event output ports (IEOPs) and restricts models to communicate through IEOPs, which have no blocking wait time on them in default configuration.

Accounting for Interpolated Event Input and Output Ports

$$\begin{aligned}\Delta T_{wall,\ \Delta T_{sim}} = \sum_{j=0}^{J} \sum_{i=0}^{\Delta T_{sim}/\Delta IE_{input,j}} & [\Upsilon_{IEIP}(ie_{i,j} \in IE_{j,\Delta T_{sim}}) + \Upsilon_{wait}(ie_{i+1,j}) \\ & + \sum_{l=0}^{K} \Upsilon_{STF}(e_l \in ie_{i,j}, S)] + \sum_{i=0}^{L} \Upsilon_{STF}(e_i \in E_{internal,\Delta T_{sim}}, S) \\ & + \sum_{i=0}^{M} \Upsilon_{STF}(e_i \in E_{prescheduled,\Delta T_{sim}}, S) + \sum_{i=0}^{N} \Upsilon_{\lambda_i}(S) \\ & + \sum_{j=0}^{O} [\sum_{i=0}^{\Delta T_{sim}/\Delta IE_{output,j}} \Upsilon_{IEOP}(ie_{i,j} \in IE_{output,j,\Delta T_{sim}}) \\ & + \sum_{k=0}^{P} \Upsilon_{IEOP}(e_k \in IE_{corrected\ outputs,j,\ \Delta T_{sim}})]\end{aligned}$$

(Eq. 1.5)

J = the number of interpolated event input ports

K = the number of conventional events generated by ie_i

L = the number of internal events generated during ΔT_{sim}

M = the number of prescheduled events during ΔT_{sim}

N = the number of times output function λ is evaluated in ΔT_{sim}

O = the number of interpolated event output ports

P = the number of corrected interpolated events if the port is in tracking mode

$\Upsilon_{IEOP}(ie_{i,j} \in IE_{output,j,\Delta T_{sim}})$ = the wall cost time of servicing the interpolated event output port for output $ie_{i,j}$

$IE_{output,j,\Delta T_{sim}}$ = the set of output *ie's* posted on IEOP port j

$IE_{corrected,j,\Delta T_{sim}}$ = the set of corrected output *ie's* posted on IEOP port j if in tracking mode

$\Delta IE_{input,j}$ = the interpolated event port input duration on input port j

$\Delta IE_{output,j}$ = the interpolated event port output duration on output port j

If each IEOP is in sampling mode, there are no corrected output events, so $P = 0$. There is also no wait time on sending output IEs, just the port servicing time Υ_{IEOP}.

Expanding $\Upsilon_{wait}(ie)$

We see that the wall clock time simulation cost is dependent partly on the $\Upsilon_{wait}(ie_{i+1,j})$ wall cost time of the next interpolated event on each IE input port j. $\Upsilon_{wait}(ie_{i+1,j})$ is composed of the wall clock time cost of transmitting the IE, the cost

$\Upsilon_{TX}(ie)$, which is a function of the IE transmit method and compute resources. It may be network delay dependent. $\Upsilon_{wait}(ie_{i+1,j})$ is also related to the cost of the input process to advance ΔIE units of time. That cost is $\Delta T_{wall,\ \Delta T_{sim}}$ for the input process, where ΔT_{sim} is ΔIE for the output port of the process. We expand the notation $\Delta T_{wall,\ \Delta T_{sim}}$ to be specific for a process. Let $\Delta T^{p}_{wall,\ \Delta T_{sim}}$ be the wall clock simulation time cost of process p advancing ΔT_{sim} units of simulation time. Therefore, $\gamma^{p'}_{wait}(ie) = \Delta T^{p}_{wall,\ \Delta ie_{sim}} + \Upsilon_{TX}(ie_p)$, for input process p sending an ie_p to process p'.

Minimum $\Delta T_{wall,\ \Delta T_{sim}}$ time for a parallel simulation

A parallel simulation consists of a finite set of P processes p_i, each connected through Interpolated Event Input and Output Ports. We can construct a directed graph of the system connections through IE ports. Of interest is determining the reduction of $\Delta T^{P}_{wall,\ \Delta T_{sim}}$, the wall clock time of the entire set of processes to advance ΔT_{sim} simulation units as a result of running the simulation in parallel. For the system to simulate a time segment ΔT_{sim}, each process must advance ΔT_{sim} simulation units by virtue of the local causality constraint. Since each process runs in parallel, *the system simulation runs no faster than the process which takes the longest wall clock time to advance a simulated time segment of ΔT_{sim} units.* This observation is important for determining relationships between IE resolution and parallel speedup.

The $\Delta T_{wall,\ \Delta T_{sim}}$ and ΔIE relationship

We can gain some insight into the relationship between $\Delta T^{P}_{wall,\ \Delta T_{sim}}$ for a simulation of P processes and the choice of ΔIE for each port. Consider the following simplifications: the service time of each IE ($\Upsilon_{IEIP/IEOP}$) port is negligible, each process has one IE input and one IE output port, and the system output function λ is only evaluated at every IE input event. Further, there are no internal events, prescheduled time points, or re-sourced events on each input port. The only varying parameters are

$\Upsilon_{wait}(ie_{i+1,j})$ and $\Upsilon_{STF}(e_l \in IE_{i,j}, S)$, that is, the wait time on the input IE and the system transfer function cost time of processing one event created by $ie_{i,j}$. Let $\Delta T_{sim} = \Delta IE$ be constant for each port. We are simply evaluating the time for all of the processes to advance one $\Delta T_{sim} = \Delta IE$ time unit. Since they each must advance this unit in parallel, and all must advance this unit, we have $\Delta T^{P}_{wall,\ \Delta T_{sim}} \geq \max_{p=0\ldots|P|}[\Upsilon^{p}_{wait}(ie_{i+1,j}) + \Upsilon^{p}_{STF}(e_l \in IE_{i,j}, S)]$. This is the maximum wall clock cost of each process to receive one IE and evaluate the state transition function for it. For each process p', $\gamma^{p'}_{wait}(ie) = \Delta T^{p}_{wall,\ \Delta ie_{sim}} + \Upsilon_{TX}(ie_p)$, that is, the wall clock wait time for the input process to produce the IE and the wall clock cost of sending the ie_p, $\Upsilon_{TX}(ie_p)$. Each process produces one IE and processes one STF transition.

If we allow $\Upsilon_{TX}(ie_p)$ to be a constant $\Upsilon_{TX}(ie)$ for each process, the simplified conditions equation becomes:

$$\Delta T^{P}_{wall,\ \Delta T_{sim}} \geq (\max_{p=0\ldots|P|}[\Upsilon^{p}_{STF}(e \in IE, S)]) + |P| \times \Upsilon_{TX}(ie)$$

(Eq. 1.6)

If each $\Upsilon^{p}_{STF}(e \in IE, S)$ can be approximated by a constant $\Upsilon^{p}_{STF}(e \in IE, S) = k$, then:

$$\Delta T^{P}_{wall,\ \Delta T_{sim}} \geq (\Upsilon^{p}_{STF}(e \in E, S)) + |P| \times \Upsilon_{TX}(ie)$$

(Eq. 1.7)

This is simply the cost of progressing one *STF* each process plus the cost of sending $|P|$ *ie* messages. Several tradeoffs emerge. First, $|P| \times \Upsilon_{TX}(ie)$ increases linearly as parallelism increases (more processes are added) if $\Upsilon_{TX}(ie)$ is invariant to the number of *ie* messages sent in the system. In practice, $\Upsilon_{TX}(ie)$ increases on message volume, is dependent on network delay, and is dynamic. Also, in practice, $\Upsilon^{p}_{STF}(e \in$

$IE, S)$ increases on the number of states and is dependent on computational performance and resources at the time of evaluation.

But a simple relationship emerges: increasing messages increases simulation wall clock time, and increasing a process state complexity increases simulation wall clock time. If we reduce $Y_{STF}^{p}(e \in IE, S)$ by dividing the state of one process in the set into two processes (adding one more process to the set a result) we increase the coupling cost simulation of by adding $|P+1| \times Y_{TX}(ie)$ messages. The partitioning of state is only successful if $\max_{p=0\ldots|P+1|}\left[Y_{STF}^{p}(e \in IE, S)\right] < |P+1| \times Y_{TX}(ie)$. That is, the reduction of the maximum $Y_{STF}^{p}(e \in IE, S)$ by adding a new process (and dispersing state) is less than the cost of sending more messages, $|P+1| \times Y_{TX}(ie)$. If we further account for multiple IE ports and static ΔIE configuration for each port:

$$\Delta T_{wall,\ \Delta T_{sim}}^{P} \geq \max_{p=0\ldots|P|}\left[\sum_{j=0}^{J}\ \sum_{i=0}^{\Delta T_{sim}/\Delta IE_{i,j}} Y_{STF}^{p}(e_i \in IE_i, S)\right] + \left(\sum_{p=0}^{P}\sum_{k=0}^{K_p} \Delta T_{sim}/\Delta IE_{p,k}\right) \times Y_{TX}(ie)$$
$$(number\ of\ IE\ messages)$$

(Eq. 1.8)

In Equation 1.8, $\Delta IE_{p,k}$ refers to ΔIE configuration for output port *k* of process *p,* K_p refers to the total number of output ports for process *p*, and *P* refers to the total number of processes. Also, in Equation 1.8, the wall clock cost of process *p* sending an IE is $Y_{TX}(ie)$ and assumed constant across processes for simplification. Another tradeoff emerges from Equation 1.8. Each port may have different resolution (ΔIE). Driving a low $Y_{STF}^{p}(e_i \in IE_i, S)$ process with many IEs may cost more than driving a high

$Y_{STF}^{p}(e_i \in IE_i, S)$ process with fewer IEs. Increasing resolution adds more $Y_{TX}(ie)$ cost (more messages) in both cases. Let process p be the process with highest cost: $\max_{p=0\ldots|P|}\left[\sum_{j=0}^{J}\sum_{i=0}^{\Delta T_{sim}/\Delta IE_{i,j}} Y_{STF}^{p}(e_i \in IE_i, S)\right]$. Increasing resolution will strictly increase this cost (more ports or smaller ΔIE on that process is introduced). If we pick a lower cost process p', and increase input resolution to ΔIE' on that process (or add more ports), then the event handling time cost for that process increases to $\sum_{j=0}^{J}\sum_{i=0}^{\Delta T_{sim}/\Delta IE'_{i,j}} Y_{STF}^{p'}(e_i \in IE_i, S)$. The net simulation cost bound does not change if:

$$\sum_{j=0}^{J}\sum_{i=0}^{\Delta T_{sim}/\Delta IE'_{i,j}} Y_{STF}^{p'}(e_i \in IE_i, S) + \underbrace{\left(\sum_{p=0}^{P}\sum_{k=0}^{K_p}\Delta T_{sim}/\Delta IE_{p,k}\right)}_{(number\ of\ new\ IE\ messages)} \times Y_{TX}(ie)$$

$$\leq \max_{p=0\ldots|P|}\left[\sum_{j=0}^{J}\sum_{i=0}^{\Delta T_{sim}/\Delta IE_{i,j}} Y_{STF}^{p}(e_i \in IE_i, S)\right] + \underbrace{\left(\sum_{p=0}^{P}\sum_{k=0}^{K_p}\Delta T_{sim}/\Delta IE_{p,k}\right)}_{(number\ of original\ IE\ messages)} \times Y_{TX}(ie)$$

(Eq. 1.9)

Although the summation notation looks the same for the number of IE messages on each side of the equation, the $\Delta IE_{p,k}$ resolution is different for process p' under increased resolution (ΔIE') of the left side of the equation compared to the right side. Simplifying, if ΔIE is constant for each process, each process has one input and one output port, and $Y_{STF}^{P}(e \in E, S)$ is constant or maximum over the processes, then we can derive a speed/resolution tradeoff. Equation 1.8 reduces to:

$$\Delta T^{P}_{wall,\ \Delta T_{sim},\ \Delta IE} \geq {}^{\Delta T_{sim}}/_{\Delta IE}\,(\Upsilon^{P}_{STF}(e\ \in IE, S)) + {}^{\Delta T_{sim}}/_{\Delta IE} \times |P| \times \Upsilon_{TX}(ie)$$

(Eq. 1.10)

If we reduce resolution to ΔIE' > ΔIE for speedup (fewer messages, but reduced resolution over the same ΔT_{sim}), then:

$$\Delta T^{P}_{wall,\ \Delta T_{sim}, \Delta IE'} \geq {}^{\Delta T_{sim}}/_{\Delta IE'}\,(\Upsilon^{P}_{STF}(e\ \in IE, S)) + {}^{\Delta T_{sim}}/_{\Delta IE'} \times |P| \times \Upsilon_{TX}(ie)$$

(Eq. 1.11)

The speedup ratio ($\frac{\Delta T^{P}_{wall,\ \Delta T_{sim}, \Delta IE'}}{\Delta T^{P}_{wall,\ \Delta T_{sim},\ \Delta IE}}$), after simplification becomes:

$$\left(\frac{\Delta T^{P}_{wall,\ \Delta T_{sim}, \Delta IE'}}{\Delta T^{P}_{wall,\ \Delta T_{sim},\ \Delta IE}}\right) \geq \frac{{}^{\Delta T_{sim}}/_{\Delta IE'}\left(\Upsilon^{P}_{STF}(e\ \in IE, S)\right) + {}^{\Delta T_{sim}}/_{\Delta IE'} \times |P| \times \Upsilon_{TX}(ie)}{{}^{\Delta T_{sim}}/_{\Delta IE}\left(\Upsilon^{P}_{STF}(e\ \in IE, S)\right) + {}^{\Delta T_{sim}}/_{\Delta IE} \times |P| \times \Upsilon_{TX}(ie)}$$
$$= \frac{{}^{\Delta T_{sim}}/_{\Delta IE'}}{{}^{\Delta T_{sim}}/_{\Delta IE}} = \frac{\Delta IE}{\Delta IE'}$$

(Eq. 1.12)

This is a convenient relationship. If we decrease resolution, ΔIE' = 2ΔIE, then the speedup ratio is 2, or the simulation takes at least ½ as long. If we increase resolution, ΔIE' =1/2 ΔIE, then the simulation takes at least twice as long. This assumes the following: $\Upsilon_{TX}(ie)$ is constant for each message and does not increase on the total number of messages. Next, it assumes $\Upsilon^{P}_{STF}(e\ \in IE, S)$ only evaluates on input events (no prescheduled time points or internal events) and is constant or bounded across the processes. Finally, every process has the same number of ports, each configured to the same ΔIE duration, and the time cost of servicing each port is negligible.

If $\Upsilon^{P}_{STF}(e\ \in IE, S)$ varies across processes or is bounded, and variable IE ports and IE durations exist, then $\left(\frac{\Delta T^{P}_{wall,\ \Delta T_{sim}, \Delta IE'}}{\Delta T^{P}_{wall,\ \Delta T_{sim},\ \Delta IE}}\right)$ must incorporate a detailed expansion

which accounts for each variation. This can be accomplished by the implementation software presented (Chapter Five), but for now a notated expansion occludes the important details of parallelism and coupling costs. Equation 1.8 shows that the wall clock costs of parallelism are bound by the state transition function computation time for each process, or bounded by the cost of transmitting more messages. The state transition function cost may be reduced by reducing the state size (by assigning it to more processes), but the resulting increase of messages may not result in a total system simulation wall clock time reduction. These tradeoffs are evident in the Spice acceleration experiments in Chapter Six. Eventually in the tradeoff, as parallelism increases and resolution increases, the cost of messaging increases. There exists therefore an optimum balance between resolution, number of processes, and the wall clock time for each process to evaluate its state transition function.

3.4 Error and Accuracy with IEs and IE Ports

We should seek a relationship between the resolution, which affects speed of simulation, and the accuracy of the simulation. Lower resolution may or may not be less accurate. Accuracy requires a means to measure error in the simulation. The IEOP/IEIP port restriction introduces an *error of coupling*. The IE is a zero-order sample-and-hold on the port events, and it is a piecewise constant interpolation, which can introduce sampling and delay errors. Categories of error conditions can be identified based on the potential delay and resolution flexibility of IEs. We first examine event delay.

Event Delay

Consider a process partitioned into multiple processes to reduce single process state or because of applied partitioning restrictions. The processes must communicate through events, which must traverse IE ports. Let the "distance" between two processes

be *d*, the number of "hops" (logical processes) through which the event must propagate to travel from process *p* to process *q*. It must at minimum traverse *d-1* IEOPs and *d-1* IEIPs. A critical factor is whether a process in the chain can produce an IE at the same simulated time that it consumes one. Conventionally this is rejected in discrete-event simulation programming, but we allow it here. We use the definitions from [51]: a *strongly causal* process has an output that is a function of the present state and *past* inputs only. A *weakly causal process* has an output that is function of present state and present input as well as past inputs. A strongly causal process has non-zero lookahead [51]. A weakly causal process has zero-lookahead for some inputs. The port-introduced delay for a strongly causal process is the maximum of k (the lookahead) or ΔIE on the process output port for the event. If the process is weakly causal, there may be no output port delay, but there is no guarantee the weakly-causal simulator will process the effect of the input event and send an IE on an output port in zero time. Let the path from p to q be labeled as follows: $p \xrightarrow{a} q = \{p_i \in P \mid p_i \, constructs \, a \, path \, from \, p \, to \, q\}$. The maximum simulation delay in event *e* traversing from p to q is:

$$\Delta IE^p + \sum_{i=0}^{P'} \max\left[k^i, \Delta IE^i\right], such\ that\ P' \leq \left|p \xrightarrow{a} q\right| - 1$$

(Eq. 1.13)

k^i = the lookahead for process i in the path

ΔIE^p = ΔIE of the output port of the processes relaying the event

P' = the number of processes in the path from p to q with non-zero lookahead

ΔIE^i = the ΔIE configuration for process i in the path

Since a strongly causal processes can produce events no faster than its lookahead value, the highest event resolution an IEOP can apply to that process is ΔIE = *k*. If ΔIE > *k* and the IEOP is in sampling mode, then the event will be delayed by at most ΔIE > *k* after the process produces the event.

We see that decreasing resolution (ΔIE' > ΔIE) for some processes along the chain from *p* to *q*, while decreasing simulation wall clock time, can increase the event delay in information getting from *p* to *q* in simulated time. In the extreme case, this can create a *causality error*.

Delay Introduced Causality Error

If there are two paths for two successive events to travel from *p* to *q*, then causality of the events arriving at *q* depends on the number of hops on each path and the ΔIE tuning on them. We define an expression for path delay as follows.

Let $\delta\left(p \xrightarrow{a} q\right) = \Delta IE^{p} + \sum_{i=0}^{p'} \max_{p_i}\left[k^{i},\ \Delta IE^{i}\right]$, such that p_i is in the path $p \xrightarrow{a} q$ and relays the event. Let there be a second path $p \xrightarrow{b} q$ with a different delay, $\delta\left(p \xrightarrow{b} q\right)$. Assume, without loss of generality, $\delta\left(p \xrightarrow{a} q\right) \leq \delta\left(p \xrightarrow{b} q\right)$. Let *p* send an ie_0 at time *t* along path $p \xrightarrow{a} q$, and an ie_1 along path $p \xrightarrow{b} q$ at time $t+k_p$, where k_p is the lookahead of process *p*. Since ie_0 happens before ie_1 leaving *p*, and process *q* receives ie_0 at time $t+\delta\left(p \xrightarrow{a} q\right)$ along $p \xrightarrow{a} q$, process *q* still "sees" ie_1 happens after ie_0 because it receives ie_1 at time $t + k_p + \delta\left(p \xrightarrow{b} q\right)$ along $p \xrightarrow{b} q$, and $\delta\left(p \xrightarrow{b} q\right) \geq \delta\left(p \xrightarrow{a} q\right)$. This preserves causality (ie_0 happening first from *q*'s perspective). However, if we change the ΔIE along *a*, we can add more delay. Suppose we decrease resolution, and for some process p_i along *a*, we change $\Delta IE_i' > \Delta IE_i$. If the new $\delta'\left(p \xrightarrow{a} q\right) > \delta\left(p \xrightarrow{b} q\right) - \delta\left(p \xrightarrow{a} q\right) + k_p$, then process *q* will see event ie_1 before event ie_0. This is a causality error! In general, a causality hazard is created when there are two or more paths $p \xrightarrow{i} q$, and the ΔIE_i tuning along anyone of them is greater than the lookahead for the process in $p \xrightarrow{i} q$

outputting *ie*'s at rate $\Delta IE_i > k_i$, the lookahead at process *i*. Let the "natural delay" from $p \xrightarrow{i} q$ be $\varepsilon\left(p \xrightarrow{a} q\right) = \sum_{i=0}^{P'} k_i \; such\; that\; \{p_i \in P \mid p_i \; on\; the\; path\; p \xrightarrow{a} q\}$. This is the lookahead delay for all the processes on *a*. If $\delta\left(p \xrightarrow{a} q\right) > \varepsilon\left(p \xrightarrow{a} q\right)$, then a causality hazard is opened if $\delta\left(p \xrightarrow{a} q\right) >$ any $\delta\left(p \xrightarrow{b} q\right)$ for which $\varepsilon\left(p \xrightarrow{a} q\right) < \delta\left(p \xrightarrow{b} q\right)$. That is, $p \xrightarrow{a} q$ has now become a higher delay path than $p \xrightarrow{b} q$, and $\varepsilon\left(p \xrightarrow{a} q\right) < \varepsilon\left(p \xrightarrow{b} q\right)$. This is a coupling-introduced causality hazard completely due to $\Delta IE_i > k_i$ delay. It does not guarantee a hazard will occur, but it opens the possibility for it. If *p* sends an "on" message along *a*, then an "off" message along *b*, if $\delta\left(p \xrightarrow{a} q\right)$ becomes greater than $\delta\left(p \xrightarrow{b} q\right)$ after reducing resolution on $p \xrightarrow{a} q$, then *q* will receive the "off" message first, where before the resolution change *q* would see "on" before "off" as long as $\delta\left(p \xrightarrow{b} q\right) > \delta\left(p \xrightarrow{a} q\right) \geq \varepsilon\left(p \xrightarrow{a} q\right)$.

Delay Introduced State Trajectory Error

If $\Delta IE_i > k_i$ for a process p_i in *P*, that is, the output port ΔIE configuration for a port in the process is greater than the process lookahead, then p_i may delay the output of and event *e* up to ΔIE units of simulation time if the port is in sampling mode. This introduces trajectory error in the transitions of *S*, because the receiving process of p_i will "see" the event with delay up to ΔIE units of simulation time. So the subset of *S* modeled by the receiver process will not advance to $STF(e_t)$ at event e_t, where *t* is the time of the event without IE port delay, but at time with delay up to $t + \Delta IE$. If p_i receives no other events after e_t and is *time-invariant*, then $STF(e_t) = STF(e_{t + \Delta IE})$. If a metric can be established on S_i, (the set of states modeled by process p_i), then the error in processing e_t at time $e_{t+\Delta IE}$ can be expressed. Let S_i^{t0} be the state of process p_i at time *t0*. Let S_i^{et} be the state of p_i at time e_t. Then the error of state at time *t* is $\| S_i^{et} \| - \| S_i^{t0} \|$ where $\| \|$ is the metric norm declared on points of S_i. This error will persist until time $t + \Delta IE$, at which point S_i will transition to $STF(e_{t+\Delta IE})$.

An important case where this delay does not introduce error is when the receiving process has a natural sampling function, such as modeling an analog-to-digital converter (ADC). If ΔIE is less than or equal to the sampling rate of p_i, then no delay error is introduced to the state of S_i, since $STF_i(e_t)$ will not be evaluated until the end of the sampling period of p_i. Note that a causality error may still occur if there are multiple paths to p_i, but not a ΔIE-introduced state transition delay.

Masked Event State Trajectory Error

A more serious condition is if $\Delta IE_i > k_i$ for a process p_i in P, and an IEOP of p_i is in sampling mode, and p_i evaluates two event outputs on the port during time ΔIE_i. Due to sampling mode, only the last occurring event will be transmitted through the port. The first event will be "missed," or "ΔIE-masked." This introduces a state trajectory error because the receiving process never receives the first event. In fact, p_i could completely alias a changing event segment into a constant output event (if $\Delta IE_i > k_i$). Once again, if the receiving process has a sampling rate greater than ΔIE_i, then the receiving process would not see the event anyway. The effect of masked errors may not be measurable without a metric over S, but their occurrence can certainly be counted in the simulation. If a change in ΔIE_i along some p_i in P introduces more masked event errors, the accuracy of the trajectory of S should be considered.

Sample-and-hold Error

In sampling mode, an IEOP is a zero-order sample-and-hold interpolation (ZOH) on the event segment sent through the port. Sample-and-hold delay can be analyzed with system theory if each p_i in P is a linear, time-invariant process. In this case each p_i can be modeled with a continuous time transfer function $H(s)$, and discrete time transfer function $H(z)$ for the time points p_i. Each IEOP with constant in ΔIE can be modeled as a process

with the zero-order sample-and-hold transfer function. The ZOH adds a potentially destabilizing phase lag to any feedback loops in the system connectivity.

$$H_{IEOP}(s) = \frac{1 - e^{-Ts}}{s}, \text{ such that } T = \Delta IE$$

(Eq. 1.14)

For a process *G(s)* preceded by an IEOP, the discrete-time transfer function of the IEOP and system is:

$$H_{IEOP*G(s)}(z) = (1 - z^{-1}) Z\left\{\frac{G(s)}{s}\right\}, \text{ such that } z = e^{s\Delta IE}$$

(Eq. 1.15)

The phase delay of the ZOH IEOP can create instability in feedback loops if the nodes of the loop can also be represented with linear transfer functions [37]. The ZOH adds a potentially destabilizing phase lag to any feedback and can shift poles outside of the regions of stability in the complex s-plane and the z-plane.

Depending on the loop gain and bandwidth of a feedback loop in the system, each IEOP adds a phase lag that can make the loop unstable. Franklin, et al. [37] recommend the sampling rate of a feedback loop to be 20 to 30 times the bandwidth of the loop for reliable stability and digital implementation of a continuous control system.

IE Signal Difference

Another metric of difference between two simulations where there is no metric on the system state space may be established in terms of IE stream difference from one simulation to the next. Because the KPN-IE simulation is completely determined by the IE streams of the simulation, error in the simulation from one resolution to the next may be parameterized by differences on IE streams after the stream concatenation operation is applied to a record of the simulation. If a metric between IE streams is established, such

as any function norm metric established on a function space, a difference between simulations may be expressed by the magnitude of difference in IE streams established by the function metric. The metric need not be complicated, since an IE stream is a piecewise constant function.

Simultaneous Events

The KPN-IE method does not prevent the problem of ambiguous causality due to simultaneous events, which are events with the same time stamp [36]. The behavior of a system with simultaneous events may depend on the order in which the system processes the events, which can be a function of when the system receives the events. This can be a non-deterministic property external to the simulation determined by the particular network conditions upon which the simulation is carried. Order-sensitive simultaneous events introduce the problem of *repeatability* between two simulations [36].

In addition to non-repeatability, simultaneous events and zero-lookahead models may assign causality between two events (say an input causing an output), but they will have the same time stamp due to zero-lookahead in the process. Methods for marking causality in the time stamp format field by adding additional lower-precision bits to the time stamp while still preserving the time-based ordering of the upper bits are given in [36]. The IE format does not preclude such packed time stamps, or "dense time," as described by Nutaro [51]. Rather, the IE format only requires that a total order be defined on the set T of tags from which the t_m and t_n values are taken.

Another approach to the simultaneous events problem is that each node specify an internal priority ranking of simultaneous input sources. The Parallel DEVS (PDEVS) formalism adds a selection function to the DEVS formalism to accomplish this [21]. For the cyber-physical systems studied in Chapter Six, the node-specified priority approach to simultaneous events is assumed. For simulators with closed internal architecture, but an

interfacing API sufficient to host an IEOP or IEIP, the internal node-specified priority approach to simultaneous inputs may be required if dense time is not used. If the simulator cannot guarantee a simultaneous event policy, simulations with zero-lookahead cycles with KPN-IE may not be repeatable.

For systems that iterate with zero-time messages ("delta delay") [34], KPN-IE allows sending of IEs with $t_m = t_n$, equivalent to conventional events. However, KPN-IE will not prevent deadlock in the dataflow with these conditions if there are cycles of zero-time messages. However, if every cycle in the simulator connection dataflow graph has at least one node with non-zero lookahead, then by the property of Chandy/Misra/Bryant null message argument [7], the system will not deadlock. This can be achieved by assuring that at least one IEOP port in every cycle of connected IE ports has a ΔIE configuration greater than zero.

ΔIE and a Subset of System State

If no metric exists over the entire system state *S*, but over a subset of elements of *S*, the difference between one simulation at a resolution ΔIE and a repeat of the simulation at a resolution ΔIE' can be expressed by difference in the trajectory or final state of elements in *S* that have a metric. This measure of accuracy in terms of a chosen ΔIE and the difference in trajectory over a subset of elements of *S* is applied in the Chapter Six experiments. Under one resolution ΔIE, a control condition, the observed trajectory of one element of the system state considered important is recorded during the simulation. Under a second resolution ΔIE', the simulation is repeated and the trajectory for the same element recorded. Error can be expressed in terms of difference between the final state of the element under the two resolutions, or a function metric between the two trajectories over the entire simulation.

3.5 Chapter Summary

Interpolated Events (IEs) are a novel data type introduced in this work for representing simulator signals and combining signal value information and simulator time synchronization into a single message. Operations on IEs are defined in terms of the IE signal value start and signal value expiration times captured in the IE token format. IEs enter simulator logical processes through Interpolated Event Input Ports (IEIPs) and exit through Interpolated Event Output Ports (IEOPs). IEOPs may be configured in a sampling mode for flexible signal representation or in tracking mode for exact signal representation. The chapter provides a formal specification for IEIP and IEOP behavior, which is designed to be instantiated in simulator device level interfaces.

IE ports yield an analytic expression of the wall clock time cost of simulation for a coordination of parallel logical processes in terms of IE resolution (ΔIE). Under certain restrictions, the time cost function yields a simple relationship between simulator wall clock time speed up and IE resolution, becoming simply a ratio of one IE resolution configuration to the next. Finally, IEs offer categories of error analysis of simulator signals when expressed as streams of IEs. When a logical process IE resolution is less than or equal to, and divides the logical process lookahead, IE streams may not introduce and signal coupling errors. When IE resolution is greater than logical process lookahead, or does not divide it, IE streams can mask or delay signal information in exchange for potential simulation speed up gains due to relaxed IE resolution compared to the logical process lookahead.

CHAPTER FOUR. KAHN PROCESS NETWORKS AND INTERPOLATED EVENTS

The Kahn Process Network (KPN) and Interpolated Event (IE) method (KPN-IE) enables parallel and distributed simulation (PADS) through the synchronization properties of KPN dataflow when restricted to IEs as the KPN data tokens. The IE tokens provide signal information and simulated time communication, and the KPN dynamics forward IEs through the KPN dataflow formalism from signal producers to signal consumers. The blocking properties of KPN node rules enforce the important local causality constraint of PADS upon connected simulators, and tracking of IEs in the KPN data streams yields a bounded measure on important coordinating parameters for conservative and optimistic coordination, namely, the conservative Lower Bound Time Stamp (LBTS) and optimistic Global Virtual Time (GVT) values. The coordinating properties of IEs and KPNs, embedded in their formalisms, endeavor to reduce the simulator interfacing implementation burden for connected simulators and the coordinating backplane functional complexity for a PADS solution, while still providing capabilities of leading solutions.

4.1 Kahn Process Networks

Kahn Process Networks (KPNs), named for Gilles Kahn and defined in [29], are dataflow networks with these properties:

- The KPN is a directed graph with arcs, representing point-to-point simplex FIFOs, and nodes, representing concurrent compute elements without interdependent side-effects.

- Nodes may read from input FIFOs and write to output FIFOs, but reads are blocking (the node stalls) if the FIFO is empty, while writes always succeed (the node does not stall).
- Nodes may not conditionally execute by FIFO sniffing, and FIFOs are unbounded (infinite depth).

The KPN is independent of the order of node execution if the KPN dataflow rules are followed. KPNs reduce to synchronous data flow networks if token production rates are static and known a priori [29]. KPNs, quite nicely, as shown by Gilles Kahn, are deterministic (in that the input-output relationships are independent of the scheduling of node execution) based on initial conditions if the FIFO rules are followed. The KPN rules are sufficient to solve the PADS simulator coordination problem if the KPN tokens are interpolated events, if FIFOs are simulator signal connections, and if the KPN nodes are concurrent, distributed simulators. This can simplify the simulator interface and backplane architecture functional requirements while providing a strong, analytical data-flow model (KPN). A KPN is completely determined by its node set, arc set, initial conditions, and FIFO producer rates. Illustration 6 shows the node state machine for compute nodes in the KPN. Nodes consume input tokens, compute (and possibly produce outputs), and then consume new tokens. If no new tokens are available for the node from the KPN network, the node blocks (waits and performs no computation). Applying the node compute sequence to PADS, each node in the KPN is a logical process (LP) that blocks when no new tokens (IEs) are available from the KPN.

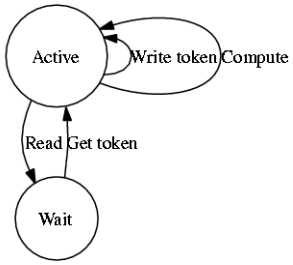


Illustration 6. The Kahn Process Network node read, wait, and execute cycle

4.2 Dynamics of Kahn Process Networks and Interpolated Events

4.2.1 KAHN PROCESS NETWORKS AND SIMULATOR COORDINATION

The Kahn Process Network restricted to IE data tokens yields a tracking of bounds on both global virtual time (GVT) for optimistic connections and the lower bound time stamp (LBTS) for conservative connections. Processes do not have to be messaged from a central controller to compute GVT or LBTS as they must in some other schemes [36]. The blocking property of KPN and IEIPs ensures a lower bound on the LBTS and GVT. These aspects will be demonstrated as we describe KPN-IE port dynamics.

The Simulator IE Port Servicing Sequence

For each node in the KPN, the node or process executes the repeated cycle given in Illustration 7. This is the simulator IE port servicing sequence, consisting of the cycle as follows.

- Read/Get input IEs
- Process until local virtual time equals IE expiration time
- Post output IEs during processing
- Re-query for new IEs at expiration time
- Block if no new IEs are available

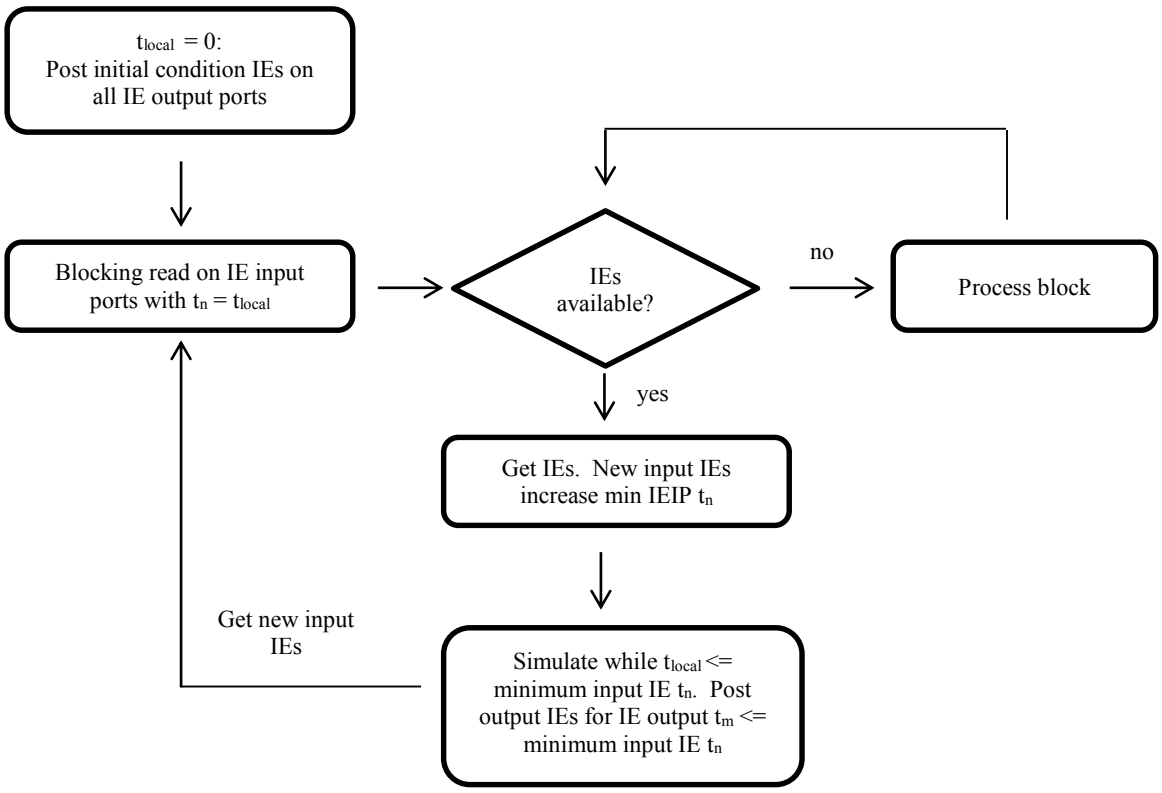


Illustration 7. The simulator IE port servicing flowchart

From Illustration 7, we see the simulation cycle of each process is determined by the IE port dynamics. The process creates the following properties.

KPN-IE Property I: A logical process cannot advance local time beyond the minimum last-received t_n value of each of the interpolated event input ports at any point in simulated time.

Proof: A logical process issues a *blocking read* on input ports for which the IE input port t_n value equals the process local time.

KPN-IE Property II: A logical process will not post an IE on any output port with a t_m value less than the minimum last sent t_n value across all IE output ports for the process for conservative processes, and the last sent t_m value for optimistic processes.

Proof: The process logical clock is non-decreasing. In sampling mode, an IE is not posted on an IE output port until the local time equals the IE port t_n value. In tracking mode, the t_m value of a corrected IE is the last sent t_m value for the port.

KPN-IE Property III: For conservatively posting IE output ports (ports in sampling mode), the lower bound time stamp (LBTS) of any future events posted by those logical process at any one point in the simulation is the minimum last-sent t_n value across all IE output ports.

Proof: By KPN-IE Property II, a conservative logical process will not post an IE on any output port with a t_m value less than the minimum last-sent t_n value across all IE output ports for the process. Let each LP keep track of the minimum last-sent t_n value across its output IE ports. The minimum t_n across each LP minimum last-sent t_n values is earliest next IE that will be sent.

KPN-IE Property IV: If every KPN IE input port is blocked at any moment in the KPN dataflow, or waiting for IEs, no new IEs will be produced in the simulation, although posted IEs may still be in transit in the network.

Proof: A blocked LP cannot produce output IEs, by node rules of a KPN.

A Conservative Coordination Example

To enhance the discussion, a conservative coordination illustration is used with three logical processes, LP0, LP1 and LP2. LP0 has one IEIP and one IEOP, LP1 has two IEIPs and one IEOP, and LP2 has two IEOPs and one IEIP. The configuration is depicted in Illustration 8.

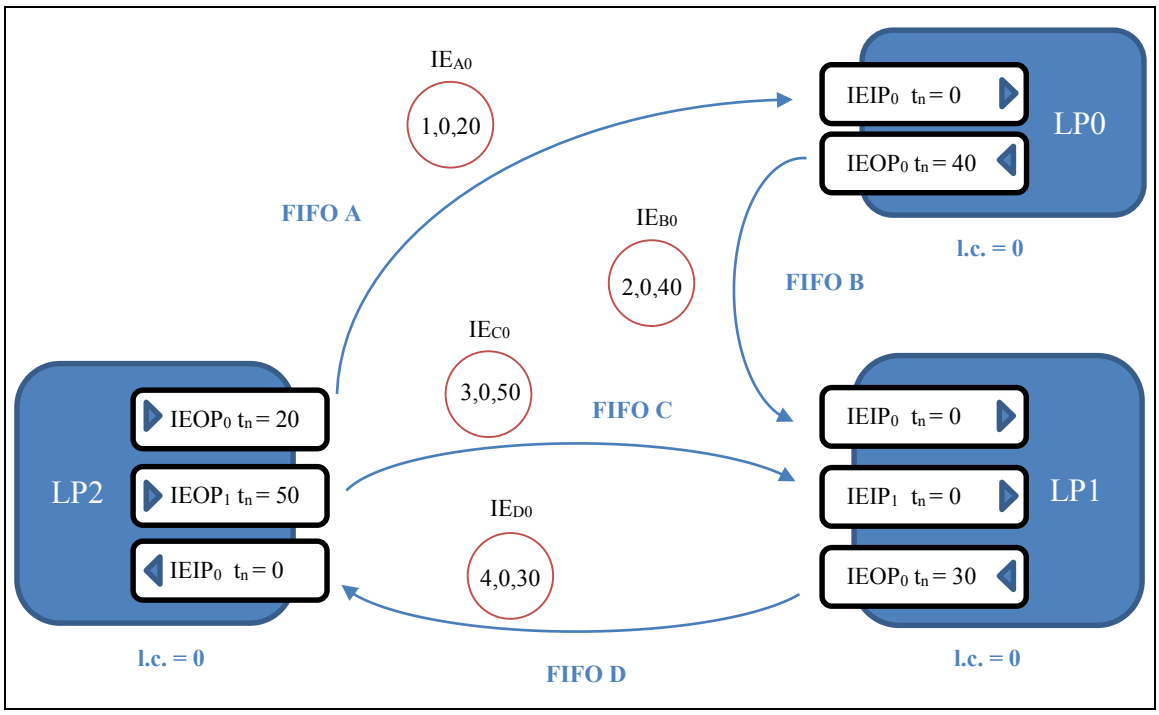


Illustration 8. Conservative KPN-IE coordination example

At simulation time zero, each LP has a local clock (l.c.) of zero and posts initial condition IEs to their IEOPs. ΔIE for LP0:IEOP$_0$ is 40, ΔIE LP1:IEOP$_0$ is 30, ΔIE LP2:IEOP$_0$ is 20, and ΔIE LP2:IEOP$_1$ is 50. The last IE posted t_n value on each IEOP is captured in the diagram. The IEs are posted to the KPN FIFOs as labeled. The IEs have values 1,2,3 and 4 respectively, and all have t_m values of 0 (initial condition IEs). Each LP is shaded blue for being in their initial condition (start of simulation states). With IE tokens in each FIFO, the LPs can now enter the running state (shaded green in Illustration 9). In Illustration 9, each LP has consumed an IE on its IEIPs, and the last received IE t_n value for each IEIP is depicted. Because each LP now has a LBTS of the minimum last received t_n value across its IEIPs, and because this value is greater than the local clock, each LP may now simulate and advance its local clock to the min last received t_n value across its IEIPs.

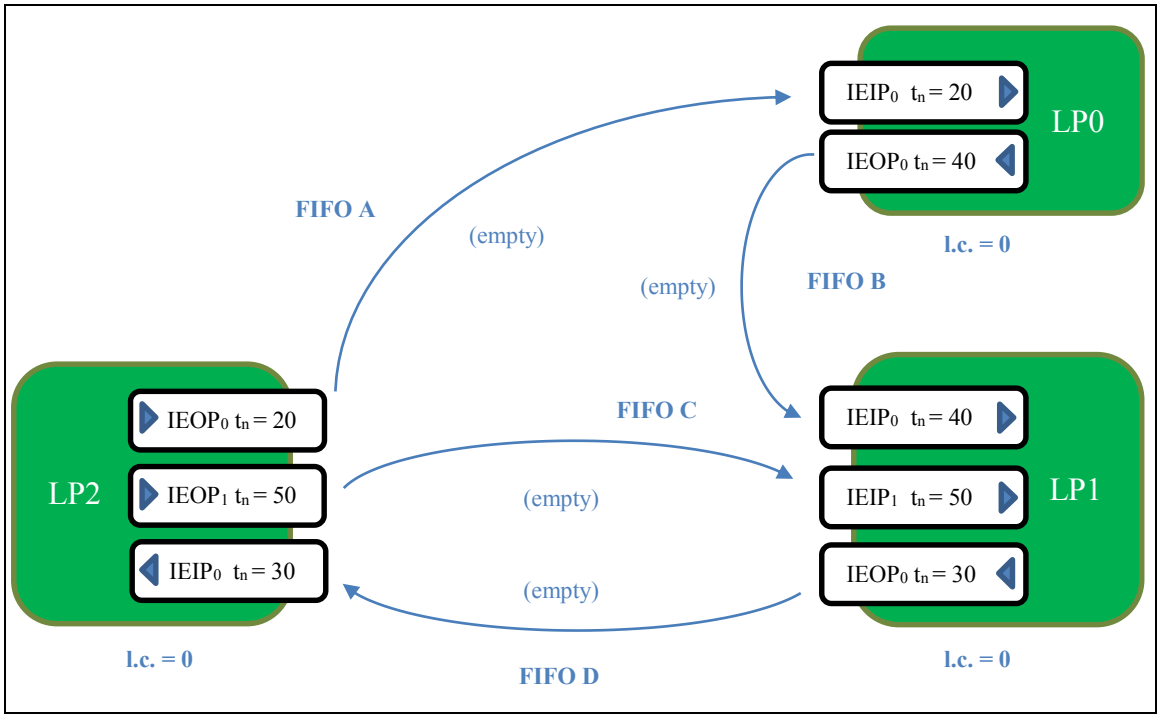


Illustration 9. Conservative KPN-IE coordination example continued 1

While simulating, each LP produces new IEs on IEOPs as shown in Illustration 10. LP2 changes its $IEOP_0$ ΔIE value to 2. In Illustration 10, the LPs have reached their LBTS across their IEIPs, that is, the local clock is equal to the minimum last received IE t_n value across the LP IEIPs. By the rules of the IEIP, the process must then query IEIPs with last received t_n values equal to the local clock (by the local causality constraint). In Illustration 10, LPs 0 and 2 are in the running state because there are IEs in their input FIFOS. LP1, however, has entered a blocked state (red), because its local clock is equal to the last received IE t_n value across its IEIPs, and there are no new IEs in its input FIFOs. When the LP1:$IEIP_0$ queries the KPN for a new IE, the process will block because the FIFO is empty for LP1:$IEIP_0$.

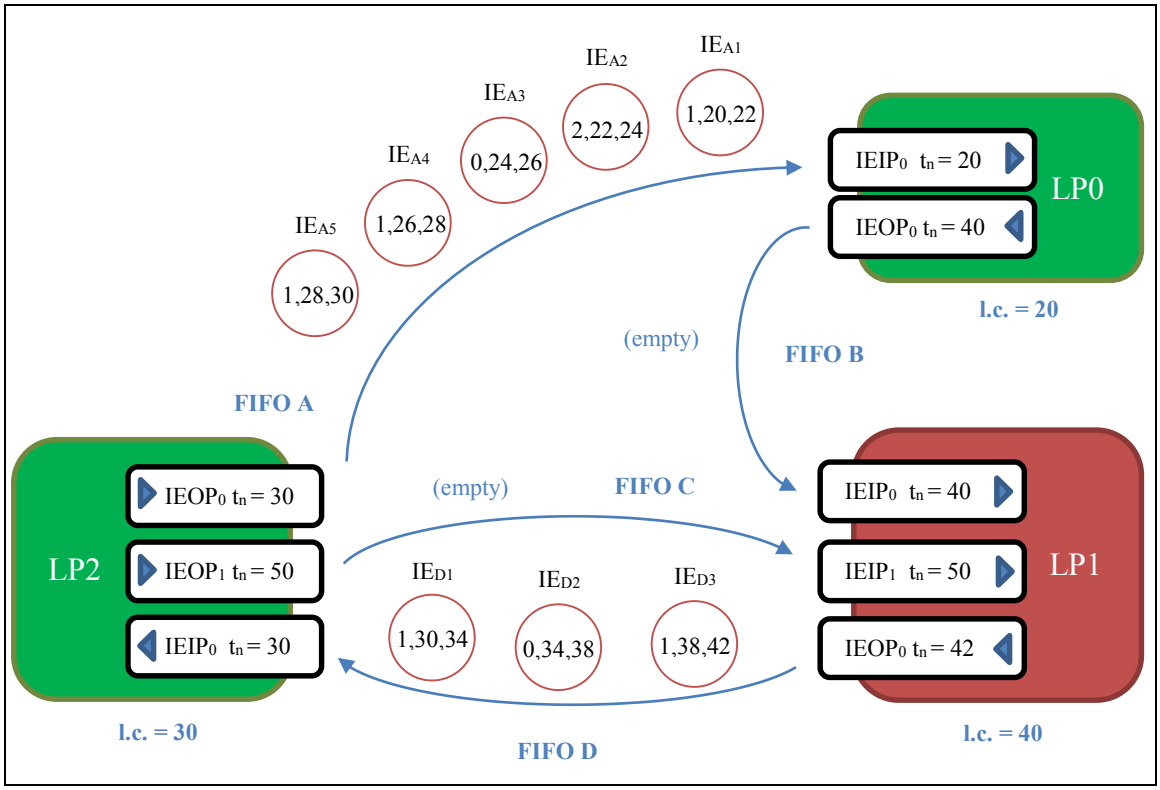


Illustration 10. Conservative KPN-IE coordination example continued 2

In Illustration 11, LP0 has consumed $IEIP_0$ input IEs, advancing its local clock to 30. LP2 has consumed input IEs in FIFO D, but blocks at local time l.c. = 42, because this is the min t_n value of the last IEs received across its IEIPs, and there are no new IEs in the KPN FIFOs C and D. So it blocks when it services LP2:$IEIP_0$. In the time of advancing its l.c. from 30 to 42, however, it has produced new IEs on its LP2:$IEOP_0$, still configured with $\Delta IE(LP2:IEOP_0) = 2$.

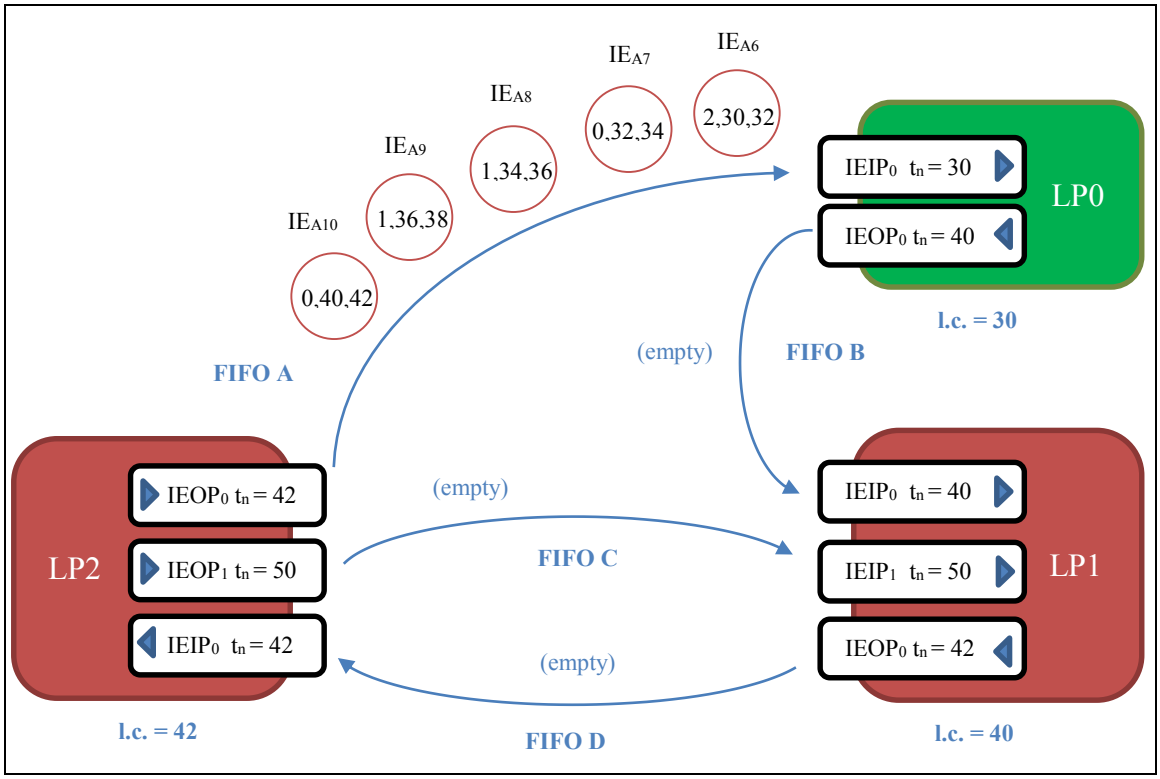


Illustration 11. Conservative KPN-IE coordination example continued 3

In Illustration 12, LP0, still in the running state, has consumed IEs in FIFO A through LP0:IEIP$_0$, and its logical clock has reached 40, the last min t_n value of IEs sent across its IEOPs. It may now produce a new output IE. It produces IE$_{B1}$, with t_n value = 80, which is the value of the t_n sent on LP0:IEOP$_0$ plus ΔIE(LP0:IEOP$_0$) = 40. With an IE now in its input FIFO, LP1 may receive the IE$_{B1}$, and enter the running state (green).

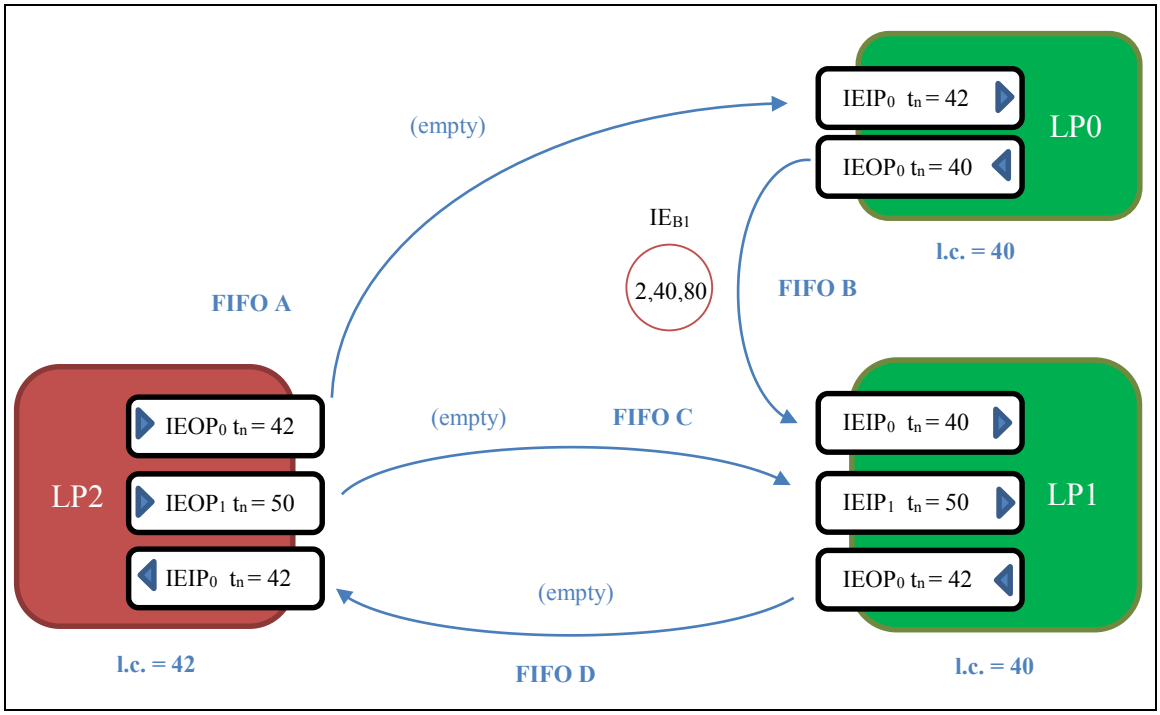


Illustration 12. Conservative KPN-IE coordination example continued 4

After running to the next min t_n received on its IEIPs, LP1 advances to local time 50, producing IEs on FIFO D in Illustration 13. This allows LP2 to enter the running state, producing new IEs and advancing its local clock in Illustration 14 to the min last sent t_n value across its IEOPs, which is 50. At this time, by the rules of the IEOP, it produces a new IE on FIFO C, allowing LP1 to enter the running state. Two properties should be evident here. First, no LP advances its local clock (l.c.) beyond the minimum last IE t_n it has received. This prevents the LP from advancing time beyond an IE it has not yet received (the local causality constraint). Next, no LP permanently blocks. This is because each IEOP has a non-zero ΔIE configuration, and by the proof of Chandy/Misra/Brant conservative LPs, no LP will permanently block if every IE connection cycle in the KPN dataflow graph has at least one LP with non-zero ΔIE (that is, at least one LP in every cycle has non-zero lookahead) [36].

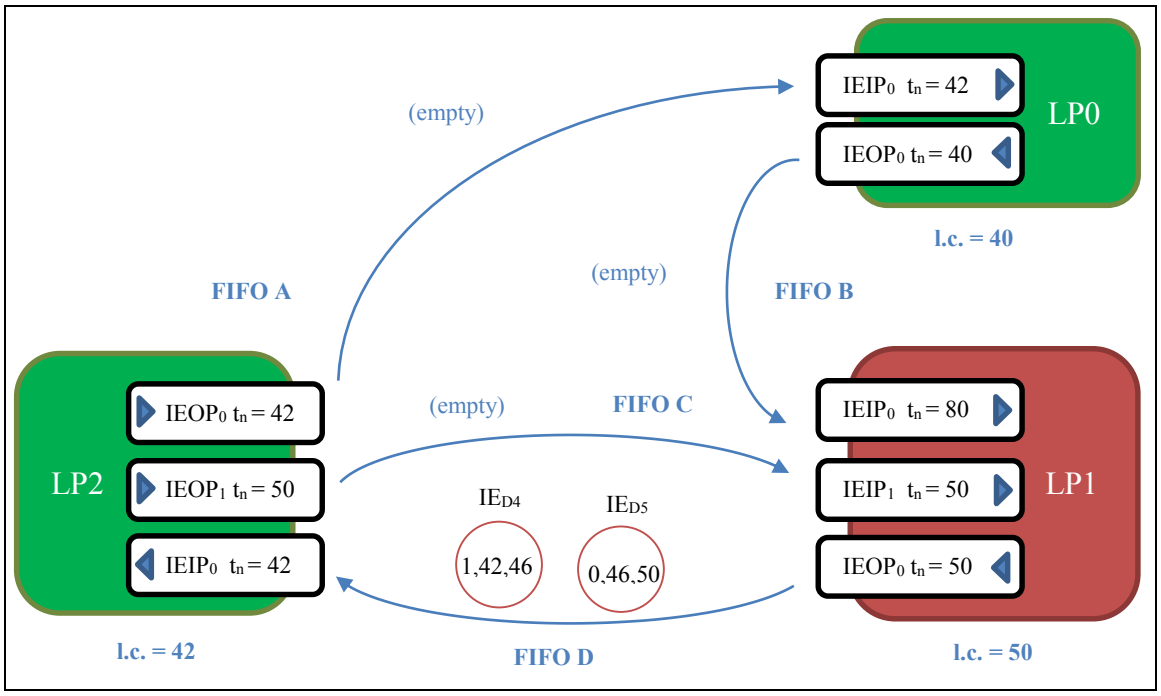


Illustration 13. Conservative KPN-IE coordination example continued 5

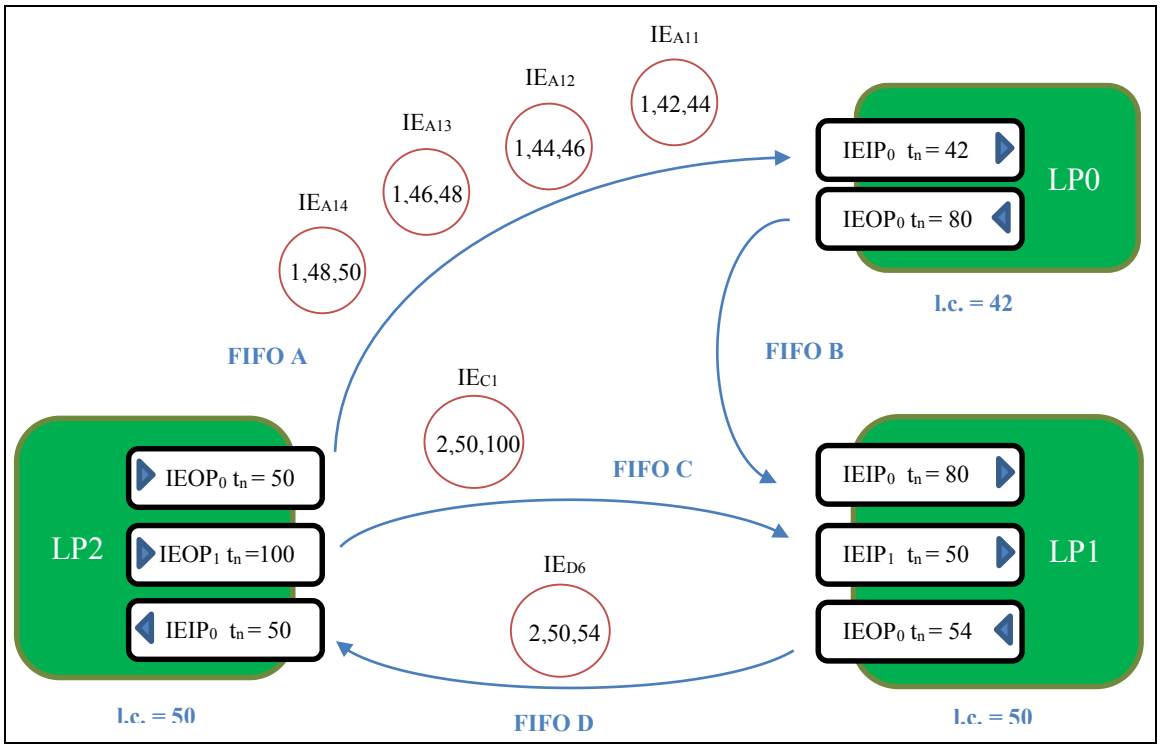


Illustration 14. Conservative KPN-IE coordination example continued 6

Important insight into the KPN-IE dynamics can be achieved if we take Illustration 14 and see it from a FIFO point of view rather than an LP point of view. Illustration 15 shows the KPN-IE network with FIFOs from their source IEOPs to their destination IEIPs.

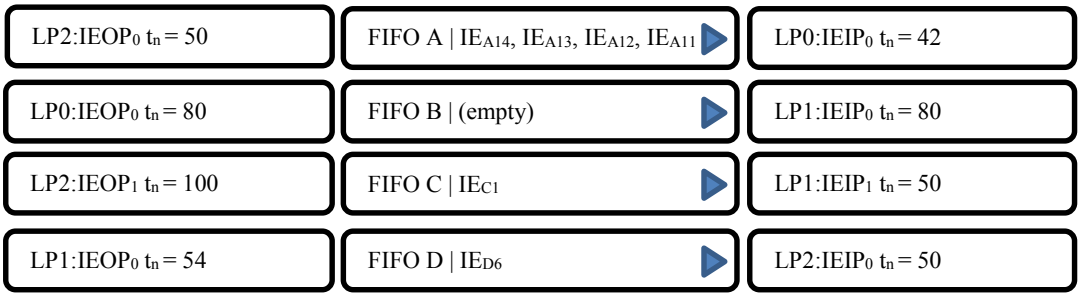


Illustration 15. Conservative KPN-IE coordination example FIFO point of view

FIFO A contains 4 IEs, FIFO B is empty, and FIFO C and D each contain one IE. The last received IE t_n value for each IEIP is given and the last posted IE output t_n value for each IEOP is given. We will see that the IE t_n values govern the KPN-IE dynamics. They are not strictly lookahead values because they will be used later to dynamically change the simulation resolution. In this conservative example, though, they represent lookahead values for each LP, but we have not been forced to send extra NULL messages per the Chandy/Misra/Bryant NULL message synchronization [36]. We have, rather, captured the information in the IE and let the KPN rules perform the blocking synchronization of the LPs. Each LP has not been required to calculate LBTS across its inputs, nor has the KPN network. The information is captured in the IE format. Synchronization happens automatically by the KPN rules and the IEIP state machine.

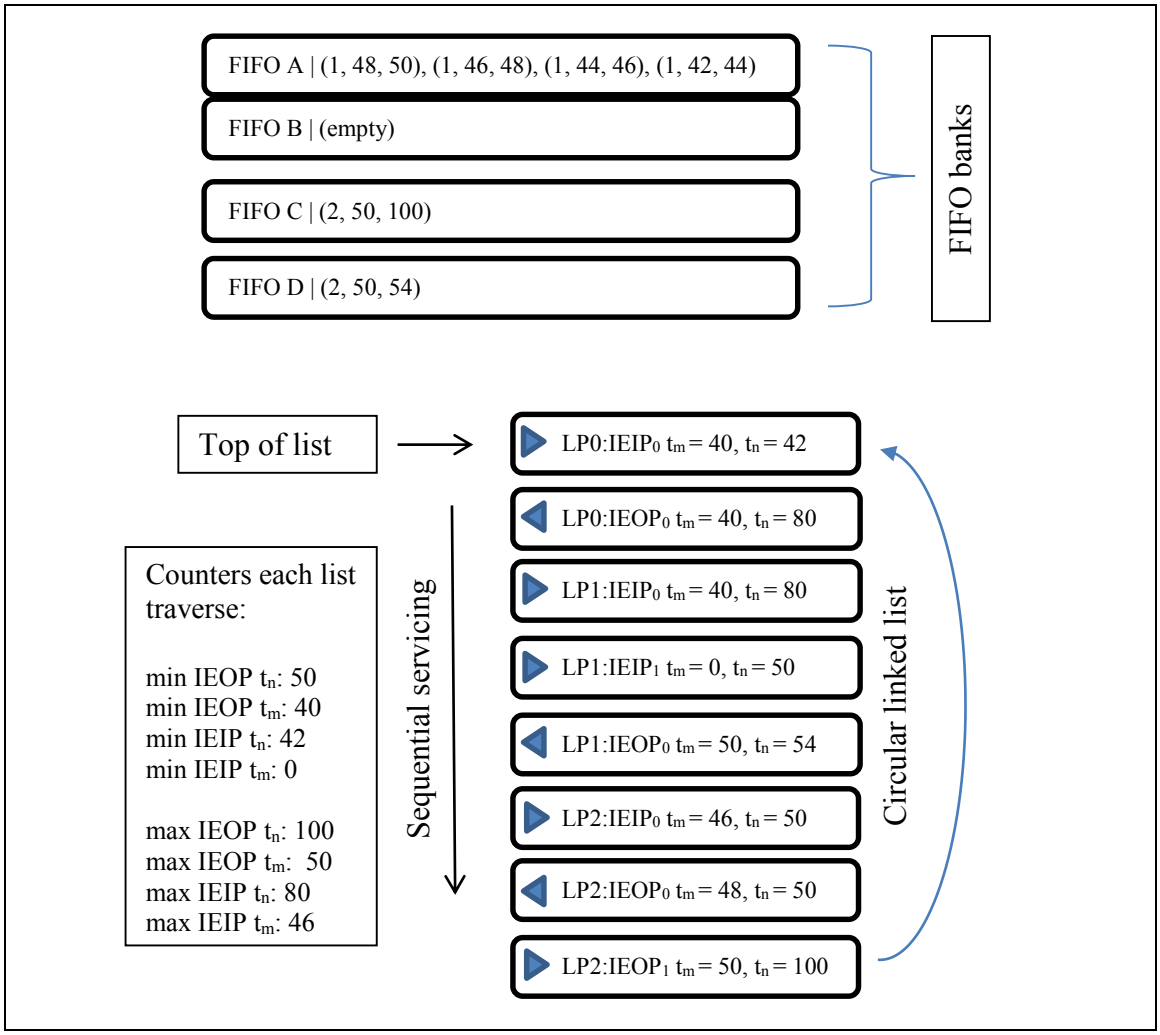


Illustration 16. Conservative KPN-IE coordination example port servicing point of view

In Illustration 16, we consider Illustration 15 from the point of view of a software agent implementing the KPN. The software agent (the simulation backplane) contains internal dynamic memory for the FIFO banks, storing IEs in them. The backplane contains a circular linked list of connection objects representing connections to LP IEIPs and IEOPs. The backplane maintains a loop of forwarding IEs from IEOPs to destination IEIPs through the FIFO banks. When the cyclic service loop services an IEOP connection, it checks to see if the connection is posting an IE. If so, it collects the IE,

stores it in its destination FIFO bank, and moves to the next connection. If the connection is an IEIP, and it is waiting for an IE, the backplane searches the FIFO banks to see if the input FIFO for the IEIP contains an IE with a t_m value equal to the last t_n value sent to the IEIP. If the IE exists in the FIFO bank, it dequeues it from the FIFO, sends it to the IEIP, and moves to the next connection. The backplane repeats this process cyclically, thereby providing the IE forwarding of the KPN network. This servicing routing is given in Illustration 17, the KPN-IE connection servicing flowchart.

In the initialization phase, the KPN backplane software agent allocates internal memory for its FIFO banks which will contain posted IEs from connections. It then listens for IE port connections (nodes which connect to the backplane through SimTalk, to be covered in Chapter Five), and organizes those connections in a circular linked list.

Several properties emerge if we update counters each time we reach the top of the connection list, as shown in Illustration 16. These counters are simply maximum and minimum values of IEOP and IEIP port properties each cycle of servicing the connection linked list. The counters are tracked as each connection is serviced and updated when the top of the list is reached each cycle. The counters capture maximum and minimum port IE t_m and t_n values across each connection. Note that no calculation or extra messaging is performed here other than counter tracking and updating.

The KPN-IE Connection Servicing Sequence

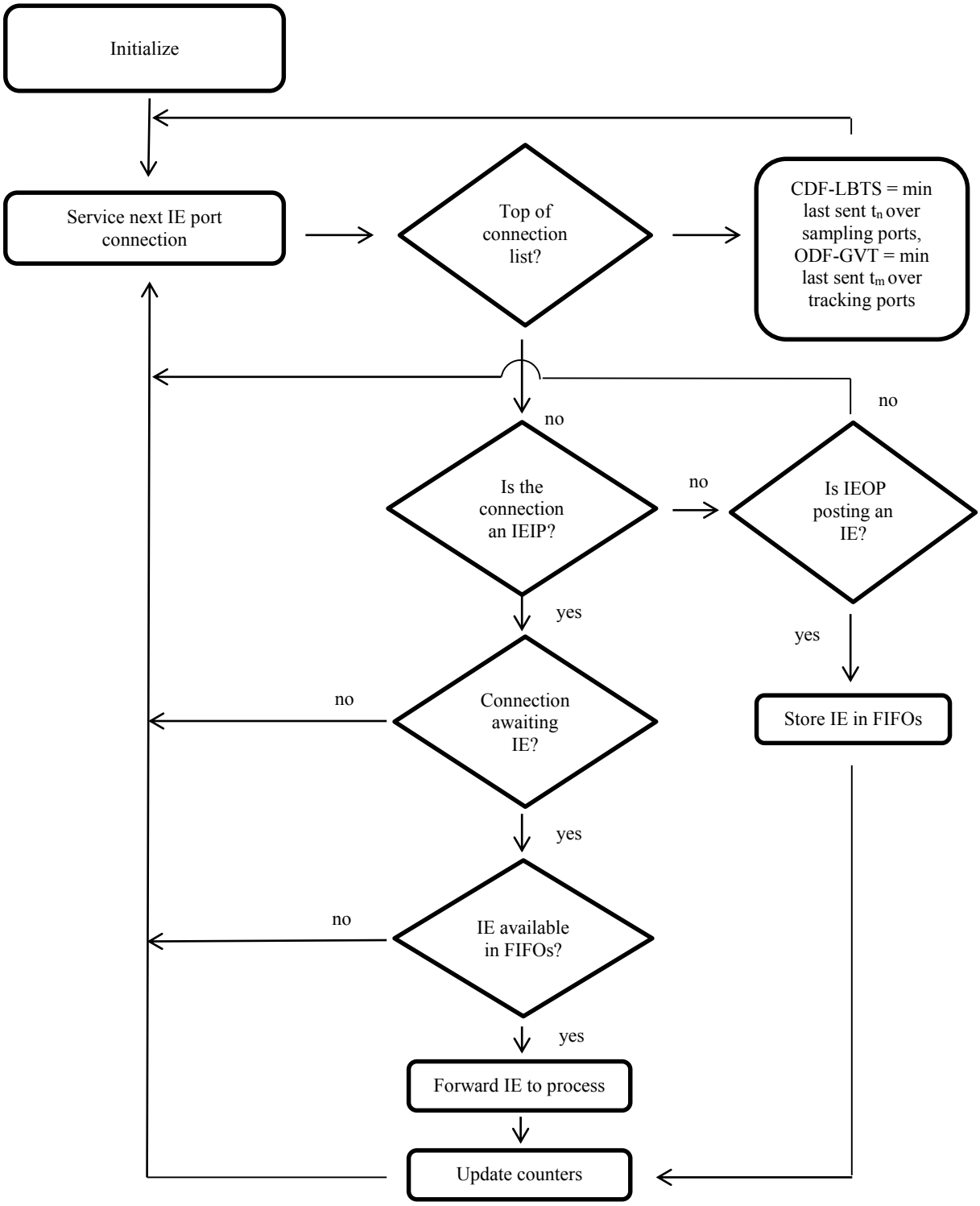


Illustration 17. The KPN-IE backplane connection servicing flowchart

4.2.2 PROPERTIES OF KPN AND IE SERVICING

By tracking the counters as given in Illustration 16 each time the connection top of the linked list is serviced, important synchronization properties emerge. These are described as follows.

KPN-IE Property V: For IE input ports in sampling mode, the $\mathbf{LBTS}_p$ at any wall clock time t is the minimum t_n value across all of the process IEIPs for a process p.

Proof: By construction of IE input ports, the process will not block and poll for a new IE until expiration time t_n on each input port. There is a minimum t_n by the ordering property on set T in IEs and the finite number of input ports. All internal events in process p are safe to process until the minimum t_n, because no new external IE will be generated with $t_m < \min t_n$ over all input ports.

Definition of $\mathbf{LBTS_{min}}$: The lower bound time stamp (LBTS) of all future messages at any wall clock time t in the simulation across all processes is called LBTS_{min}. LBTS_{min} is the min LBTS_p across all processes in the simulation at wall clock time t [36].

Definition of CDF-LBTS: The conservative data-flow lower bound time stamp (CDF-LBTS), an original term introduced by this work, is the minimum t_n value of all future IEs at any wall clock time t that any conservatively simulated (IE input ports in sampling mode) node may receive. CDF-LBTS is called the dataflow LBTS because it is measured from executing the properties of the KPN network of IE tokens.

KPN-IE Property VI: The CDF-LBTS at wall clock time t is the minimum last sent t_n value across all sampling mode IE output ports in the KPN linked list of connections at any time of measure. For sampling mode nodes, because the t_n value is non-decreasing, and because there may be messages in transit, the CDF-LBTS is less than or equal to LBTS_{min} for the network at any point of observation in the dataflow network.

Definition of ODF-GVT: The optimistic data-flow, global virtual time (ODF-GVT), an original term introduced by this work, is the minimum t_m value of all future IEs at any wall clock time t that any optimistically simulated (IE ports in tracking mode) node may receive. ODF-GVT is called the dataflow GVT because it is measured from executing the properties of the KPN network of IE tokens.

KPN-IE Property VII: The ODF-GVT at wall clock time t is the minimum last sent t_m value across all tracking-mode IE output ports in the KPN linked list of connections. For tracking mode nodes, because the last sent t_m value is non-decreasing, nodes cannot receive an IE with an earlier t_m. Therefore the GVT is the min t_m across all IE input ports. Because the ODF-GVT does not include messages in transport (which will have non-decreasing t_m values), the ODF-GVT is less than or equal to the GVT of the simulation.

Each time the KPN sequential executive services a connection, it may update the ODF-GVT and CDF-LBTS values. These are simply the minimum last sent t_m and t_n values across the list of connections. These values on non-yet serviced connections will be *non-decreasing*. So the KPN executive yields these important properties:

KPN-IE Property VIII: ODF-GVT <= GVT (optimistic simulators)

KPN-IE Property IX: CDF-LBTS <= $LBTS_{min}$ (conservative simulators)

Because conservative and optimistic coordination algorithms [36] rely on determining GVT and the $LBTS_{min}$ at any wall clock time t, the ODF-GVT and CDF-LBTS give lower bounds on these, *and do not require extra simulator messaging to calculate*. Additionally, no centralized blocking or complex algorithm is required to determine ODF-GVT and CDF-LBTS. They fall out of executing and servicing the KPN-IE dataflow network. This differentiates KPN-IE from other conservative and optimistic coordination schemes [36], which must enter barrier primitives or other

blocking primitives, calculate $LBTS_{min}$ or GVT, and communicate the value to simulators in order to process save events. KPN-IE rather allows the dataflow network to track bounds on these values in the IE content, which offers a reduction of coordination messaging at the simulation application layer based on the property that message parsing (putting additional information in a message, such as the interpolated event rather than the conventional event) can be less costly than message sending (event and time synchronization control separated in different messages).

4.3 An Optimistic Coordination Algorithm with the KPN-IE Method

Some requirements of Time-Warp [65] capable logical processes are that they maintain anti-message queues, support state-saving for rollback to earlier time points, add support for message cancellation, and perform local resource recollection [36]. These functional requirements may prevent some CPS simulators from participating in an optimistic simulation. A means is desired to support optimistic simulation but with potentially reduced functional burdens on the optimistic simulators.

With KPN-IE, lower bounds on GVT and the $LBTS_{min}$ fall out of the processing executive of the KPN without additional messaging or blocking required to the simulators. The sequence is demonstrated above in the cyclic servicing of KPN IE connections.

Furthermore, optimistic nodes must be able to roll back in time to previously executed simulation time points and states and re-execute forwardly without irreversible side effects to the simulation. Recognized coordination schemes impose additional messaging and internal queue burdens on the simulators. We advance a way to conduct optimistic simulation and only require that simulators be able to save state and roll back to a saved state on command. We impose no additional burdens on the simulators.

Let $\vec{S}$ be the last save point for each optimistic simulator. $\vec{S}$ is a vector under the Euclidean metric of time points (t_i) for which process i of the set of processes P has last saved state. $\vec{S}$ is guaranteed to exist because each processor knows its state and initial conditions at time $t = 0$ for all processes. This is the beginning save time vector.

A means is needed to tell simulators to save state. This is added to the response to read messages on simulator FIFOs from the backplane. When a simulator issues a blocking read at local time t_n, the minimum expiration time on its input ports, it requests from the KPN backplane the next IE with $t_m <= t_n$. Its local time is at t_n. When the simulator returns the IE, it prepends a "save state" message to it informing the simulator to save its current state at local time t_n before processing the new IE. If the simulator must roll back to the saved t_n, it will roll back to the time t_n right before reading the FIFO again for a new IE.

The KPN backplane tracks the time of the last save time vector, which is initially $< t_p > = 0$ for all processes p. The KPN backplane forwards IEs in normal operation, but keeps local copies of delivered IEs for t_m greater than or equal to the last time vector save for each input port connection. A correction event is detected as receiving an IE on a connection for which t_m is the same for the last IE received on that connection. This indicates an IE correction from a tracking-mode output port. The following algorithm occurs in the backplane:

If the v and t_n values of the corrected IE are the same (such as may happen in a Spice simulation rollback), the IE is dropped and the KPN continues without change. If the IE is different, either by different v or different t_n value, the simulation must be corrected. Let IE_c be the first IE received with a correction after a save point tuple $< t_i >$. First, the KPN backplane suspends IE delivery, which eventually forces each IE input port to block, by the blocking properties of the KPN. IE output port service is continued

until each output IE is delivered and the nodes are blocked. This point can be reached because no simulator will post more IEs while it is blocked on an input port read. Because in optimistic mode all posted IEs are acknowledged, we know that the all simulators blocked-condition indicates no simulators will be posting IEs. No IEs are in transit because nodes execute the KPN cycle sequentially.

When all nodes are blocked, the FIFO banks are examined for the minimum corrected IE that has been received. For tie-breaking, either the densely specified value of t from a "dense set T" [51] value is used or the receiver-ordered priority is used. Let the minimum corrected t_n value be $t_{corrected}$. This is the point before which events are safely known, but the simulators must re-simulate to this point.

Next, each FIFO bank is split on the corrected t_n value, and after the split, IEs for $t_m \geq t_n$ in the FIFOs are discarded. This discards IE and events greater in time than the corrected IE, since they are speculative. Next, a sharpen function and concatenation function are issued on each FIFO. This removes overlapping IEs (only one should exist at the corrected IE), and combines IEs for faster re-simulation. When this is finished, the corrected FIFOs have optimal and correct event information up to the correction time. Simulation until t_n then will not experience IE corrections from the last save time vector to $t_{corrected}$.

Next, each optimistic and blocked IEIP is sent a rollback message in response to its read FIFO request, rolling each simulation back to the last save point. Then, the KPN forwarding continues as usual, bringing each simulator up to time $t_{corrected}$, after which the simulation continues. Each rolled-back simulator will ask for the next IE at the rollback time for each, which is in the FIFOs because the KPN saved IEs with t_m values greater than or equal to the last state save request sent to each input port connection.

How often should simulators be commanded to save state? It depends on the FIFO memory depth of the KPN backplane and what simulation time distance is acceptable in rolling forward from a save point to a corrected IE point. The larger the distance, the longer the simulation time required for repeating corrected simulation segments. The shorter the distance, the more state saving required of optimistic simulators. The optimal tradeoff depends on the state saving cost of each simulator and the wall clock cost equation.

What is significant about this algorithm is that we have the bounded lag property of IEs in cycles due to IE t_n values, but don't have to worry about secondary rollback, anti-messages, simulator queue support for fossil collection, and other Time-Warp Logical Process (TWLP) overhead [36]. The overhead of event history is shifted to the backplane and removed from simulators. This is useful to CPS simulation, where we need to try to coordinate as many simulators as possible and minimize interfacing functional burdens. Most importantly, the simulators do not have to maintain anti-message queues and detect anti-messages, elegant though the anti-message annihilation concept is in the original Time Warp proposal [65].

4.4 A Combined Conservative and Optimistic Coordination Algorithm with the KPN-IE Method

It can be desired to coordinate simulators that are conservative with simulators that are optimistic. In this arrangement, conservative simulators must not be forwarded events or IEs that are subject to correction because they do not have the means to rollback to earlier time points and correct state. The ODF-GVT and CDF-LBTS values of the dataflow network can be used to guarantee safe IE forwarding between tracking mode and sampling mode conservative nodes.

Conservative nodes cannot receive IEs subject to correction, because the processes on the nodes may not have rollback capability. Conservative simulators using KPN-IE have a known LBTS for future events, the $LBTS_{min}$, which is greater than or equal to the CDF-LBTS as the KPN forwards IEs. So, optimistic simulators may process IEs from conservative simulators for which $t_n < LBTS_{min}$ if they exist in the FIFOs and have not been forwarded.

Similarly, conservative simulators may not receive IEs from optimistic simulators that may be later corrected. Since no rollback may occur to a time earlier than GVT (by definition), and since the ODF-GVT is less than or equal to the GVT_{sim}, it is safe to forward IEs to conservative simulators from optimistic sources for which the t_n of the forwarded IEs are less than or equal to ODF-GVT. If the consuming conservative node reads the IE, it will not advance beyond t_n, because of the blocking properties of IEs and input ports. Since the t_n is less than or equal to $GVT_{(optimistic\ simulators)}$, the conservative node will not receive an IE correction with time less than t_n, so it may safely execute.

It is therefore sufficient to therefore forward IEs from optimistic simulators to conservative processes only when $GVT \geq LBTS_{min}$. Since CDF-LBTS and ODF-GVT are updated each traversal of the list of connections in the KPN, it may be sufficient to repeat the following sequence in the KPN backplane:

While CDF-LBTS > ODF-GVT, do not forward IEs to conservative nodes while servicing the KPN list of ports. When ODF-GVT becomes ≥ CDF-LBTS, perform a split operation on conservative signal FIFO IEs at ODF-GVT. Then enable forwarding of IEs to conservative nodes for all $t_n \leq$ ODF-GVT. Conservative nodes will post new IEs and increase $LBTS_{min}$. When CDF-LBTS becomes > ODF-GVT, suspend conservative node IE forwarding again and repeat the cycle. This method should not deadlock as long as there is one non-zero lookahead conservative node in every cycle of nodes.

4.5 Chapter Summary

The KPN-IE dynamics allow measurement of bounds on important conservative and optimistic counters ($LBTS_{min}$ and GVT) without the additional messaging overhead of some centralized blocking based solutions [36]. The split and concatenation operations on IE FIFOs allow for an optimistic simulation algorithm that does not require anti-messages and other simulator overhead, potentially simplifying the interfacing burden for simulators that can support optimistic simulation.

The important t_n value of IEs forces simulators to obey the local causality constraint (LCC), because at t_n each input port will issue a blocking read to the KPN backplane for the next IE with $t_m = t_n$. In this way, no simulator may advance beyond the expiration time of its input IEs, so it cannot advance to a time ahead of not yet received events. The cost of blocking simulators and calculating $LBTS_{min}$ or GVT can add messaging and organizational burdens to simulators. With KPN-IE, bounds on these values are automatically extracted by the sequential processing and handling of connection ports through the CDF-LBTS and ODF-GVT values.

A drawback of the KPN-IE method is that IE messages must go through a central token router (the KPN backplane), which in this study is serviced sequentially. The cyclic service loop of the KPN connection handler can add an additional delay bottleneck when the number of connections becomes large. This cost can be lessened by creating multiple backplanes, each with no zero-lookahead connections between them, as illustrated in Chapter Five. However, in the class of studies in Chapter Six, the computational cost of the state transition function for simulators and the network messaging delay of IE tokens can far exceed the cyclic service loop time cost of forwarding tokens in the KPN backplane.

CHAPTER FIVE. SIMCONNECT AND SIMTALK IMPLEMENTATION

The SimConnect and SimTalk tools implement the KPN-IE method for parallel and distributed simulation (PADS) of cyber-physical systems (CPSs). SimConnect is a simulation backplane that routes IE tokens according to the rules of the Kahn Process Network, and SimTalk is a messaging protocol that enables simulators to exchange IEs with a SimConnect simulation backplane. Simulation clients connect to the SimConnect backplane in a client-server hierarchy through SimTalk and exchange IEs with the SimConnect backplane in publish-subscribe relationship. The implementation of the tools is described, including example distributed configurations, and software engineering factors are compared against two open source High Level Architecture (HLA) [19] implementations.

5.1 The KPN-IE Method with SimConnect and SimTalk

The KPN-IE method, and the SimConnect and SimTalk (SC/ST) simulation tools [10][11] implementing it, were developed as a backplane-based approach [12][13][14] to the CPS PADS simulation challenge. SimConnect and SimTalk can connect and coordinate large numbers of independent simulators running over distributed computation networks using the *interfacing* approach. A simulator may participate in a SimConnect-hosted combination of simulators if a SimTalk software connector has been written for it. The SimTalk connector enables communication from a simulator to the SimConnect backplane through the SimTalk protocol. Any two unrelated simulators may coordinate with each other if SimTalk connectors have been written for them and they support an OS-level software interface. The simulators communicate through SimTalk with the backplane and exchange signal information with the backplane in a client-server, publish-

subscribe architecture [11]. The simulators are decoupled from each other in terms of internal time and state, as required by PADS logical processes [36], requiring only a connection to the backplane. They send and receive signal information from the backplane, as opposed to directly to and from each other. The number of simulators that may communicate with a backplane is limited only by compute resources (distribution, speed, and memory), and the number of simulators for which SimTalk connectors are written. Multiple SimConnect backplanes may constitute a simulation as well, since they can also communicate through SimTalk.

The tools have been shown to coordinate heterogeneous simulators not previously connected [11] and large number of homogenous simulators for simulation speed up [17]. Speed up is obtained by increased parallelism (more simulators) for some systems, and by the dynamic runtime control ability of the SimConnect backplane [18] for others. Results from the tools [10][11][17] and [18], summarized in Chapter Six, emphasize the flexibility of SimConnect and SimTalk to enable CPS simulation, particularly as more SimTalk connectors are written for more simulators. A benefit of the KPN approach is that control of the global simulation is achieved by dataflow dynamics rather than a central controller. The backplane is not a controller, but rather a token router. If one signal producer is paused, for example, each consuming simulator of the signal blocks when it reads its input FIFO for that signal. This has a desired effect in source-based debugging of software in a co-simulation. When a software breakpoint is reached in a debugger, the architectural states (registers and memory) freeze for inspection. With SimConnect, due to KPN dataflow dynamics, any other consuming simulators also freeze, allowing inspection of system components (such as circuit levels) at the time of the breakpoint without probe interference. This occurs because FIFOs empty when a signal producer is paused, so consuming simulators therefore block. When the

breakpoint is passed, and the dataflow resumes, the consuming simulators continue with their local time preserved, without direction of a central controller. Any signal producing simulator can be paused to pause all of its consuming simulators and resume with time correctness, completely as a result of dataflow and the KPN blocking read property.

Comparison of the approach to the literature of PADS and backplane techniques is covered in [10][17]. Critically, the approach conforms to the required local causality constraint (LCC) of distributed simulation, a coordination rule that simulators must process external events in time stamp order if global event causality is to be preserved [6]. The LCC is observed by two effects. First, a simulator blocks per the rules of a KPN when an IE input FIFO is empty. Next, the simulator cannot advance in time beyond the expiration tags of IEs it has consumed on an input FIFO. Once its Local Virtual Time (LVT) [36] has reached the expiration time of the last interpolated event it consumed for that signal, it must re-query the FIFO for a new IE. If the input FIFO is empty, the simulator blocks, along with its LVT until a new IE arrives.

The approach has several advantages. First, because the interpolated event token communicates signal value, signal start and signal expiration time, it provides a schedulable, future LVT event when a simulator must re-sample its input FIFO, thus removing the time step and simulator advancement control from the responsibility of the backplane and interfacing API. The synchronization and control are captured rather in the token data and dataflow network, configured the by token update rates and IE durations of signal producers. Because these rates can be assigned statically or changed dynamically during the simulation, synchronization can be changed statically or dynamically, for simulation speed versus accuracy tradeoffs. The simulation can be conducted at high signal and time resolution (a narrow IE duration per signal), or at a coarse signal and time resolution (long duration IEs). Both resolutions are valuable.

By placing the synchronization effects in the dataflow network, where their properties can be completely monitored (versus the internal state of some simulators, which may not be monitored due to intellectual property), SimConnect and SimTalk can facilitate simulation causal debugging, mathematical analysis, replay, and signal capture (IEs streams are easily captured at runtime and copied into a database). Summarizing, SimConnect and SimTalk endeavor to reduce of the implementation challenges of alternative PADS solutions for coordinating CPS simulators by reducing the functional burden of simulator interfaces and backplane functionality. They attempt to offer simplicity and ease of adoption across engineering domains by virtue of a strict dataflow architecture and IE data types. Elements of the implementation will be described.

5.2 SimConnect

SimConnect, a simulator backplane [12], is a software server agent. Functionally, it forwards IE tokens from signal producer sockets to signal consumer sockets through internal memory FIFOs per the rules of a KPN in a cyclic service loop. The KPN nodes are connected, concurrent, and independently running simulators communicating through SimTalk with the backplane, although technically the SimConnect backplane doesn't "know" they are simulators. It only forwards IEs among connected SimTalk sockets to their destinations. A destination may be a monitoring terminal, for example, as long as it supports the SimTalk protocol.

SimConnect is currently single-threaded, but it can be upgraded to support multi-threaded servicing as high performance software server daemons do. But a CPS simulation is not restricted to one SimConnect backplane. Multiple SimConnect backplanes may be instantiated across resources, each with a client set of simulators.

SimTalk supports signal exchange from backplane to backplane, and thus from one simulator to any other in the system, as shown in Illustration 19.

5.3 SimTalk

SimTalk is a light weight message passing protocol for simulators to send and receive messages from a SimConnect backplane server. SimTalk is instantiated through SimTalk software connectors written for simulators that support an operating system-level programming interface. The messaging protocol facilitates IE signal publish and subscribe requests, read or write operations to connection FIFOs per the rules of a KPN, and communication for dynamic runtime control [17]. Implementing the SimTalk protocol can be done through any blocking, distributed message-passing API. For these studies, SimTalk is implemented through BSD/Unix socket calls with blocking reads, non-blocking writes, and ASCII string message content sent over TCP/IP for reliable delivery. A SimConnect/SimTalk hosted distributed simulation may send millions of IEs through SimTalk among connected simulators over a few seconds of simulation time, limited only by the speed of the network and distributed simulator assignment. Studies of IE count and simulation time in a distributed SimConnect/SimTalk simulation are given in [11] and [18].

The SimTalk messaging protocol defines four introductory basic messages from client to server for exchanging IEs:

Subscribe <signal name> – declare to the backplane that the client will receive interpolated events for signal <name>.

Broadcast <signal name> – declare to the backplane signal <name> and allocate a FIFO arc to store posted interpolated events for signal <name>.

Get <signal name, t> – A FIFO read operation. Get the greatest interpolated event (v, t_m, t_n) on the signal <name> FIFO such that $t_m \leq t < t_n$. The consuming simulator blocks until an IE is available from the server.

Set <signal name, v, t_m, t_n> – A FIFO write operation. Post an interpolated event (v, t_m, t_n) on signal <name> FIFO. The server delivers it to all FIFOs registered to receive interpolated events on signal <name>.

From server-to-client, there are three basic messages, supporting flow control, rollback, and dynamic resolution:

TxACK – After a signal producer posts, it can optionally wait for a server-to-client *TxACK* message indicating the posted IE has been consumed by at least one consumer node. This can serve to flow control signal producers in acyclic networks so that producing nodes do not excessively out-pace consuming nodes.

Rollback to <t> – For optimistic execution, a rollback message from server to client directs the client to rollback to an earlier local time to process a "straggling" event [10]. The node must support state saving and restoration to support rollback messages.

Resolution change at <t_k, ΔIE> – For dynamic IE signal resolution, the server can instruct the client to change an IEOP ΔIE resolution at the node's local time of t_k. For example, if a producer posts an IE (v, t_m, t_n), due to update again at time t_n, the server can send a "res <t_k, ΔIE >" message instructing the producer to post the next IE at time t_k with resolution ΔIE instead. If ΔIE > t_n - t_m, the signal resolution has been relaxed. If ΔIE < t_n - t_m, the signal resolution has been increased. Resolution change messages persist until the port internally changes the IEOP resolution or another resolution change message is received. *Any simulation agent in the simulation may send a resolution change request to*

another IEOP in the system. This a strength of the centralized backplane, that any participating agent can dynamically change simulation resolution.

5.4 Dynamic Resolution

Dynamic resolution is achieved with IEs by two means. First, a signal producer has complete write permission over an IE's posted t_n value, the expiration time of the IE (v, t_m, t_n). A signal producer may vary this value any time during the simulation based on internal knowledge of the signal's change frequency, lookahead, or some other criteria. Changing the IE t_n value will change the time that a consuming simulator of the IE next queries the backplane for that signal, thereby changing the time of next synchronization. As t_n increases beyond t_m, the IE duration increases, and therefore the event resolution relaxes (the time between synchronizations on the IE signal increases). As t_n approaches, yet still is greater than t_m, the IE duration decreases, so the event resolution increases, but more synchronization events occur.

A second method to achieve dynamic resolution with IEs is for the backplane, simulation operator, or another simulator to command a signal producer to change its [$t_n - t_m$) duration of future IEs during the simulation. In this way an agent may externally vary the IE duration of a signal producer, thereby throttling the rate of its incoming signals. External IE resolution change requests are registered in the backplane in the form of (signal name, time, resolution) 3-tuples. These can be entered at any time during the simulation through the SimTalk protocol to the backplane by any simulation participant (simulator or terminal). Application of these resolution change messages for dynamic simulation runtime control is given in [18].

5.5 Distribution, Synchronization, and CPS Simulators

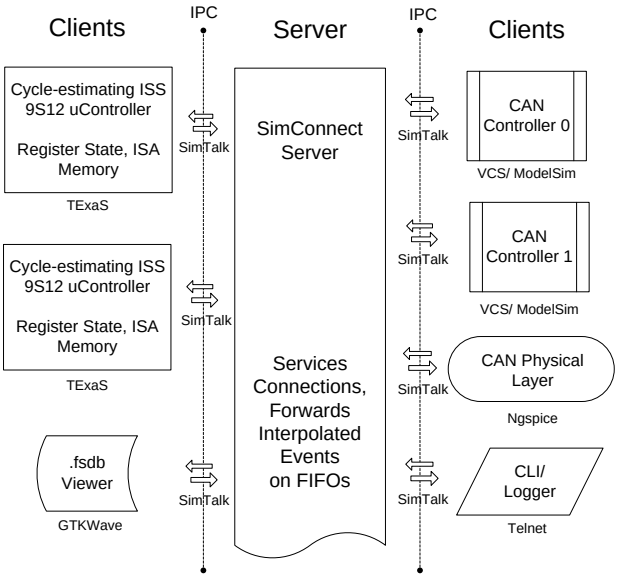
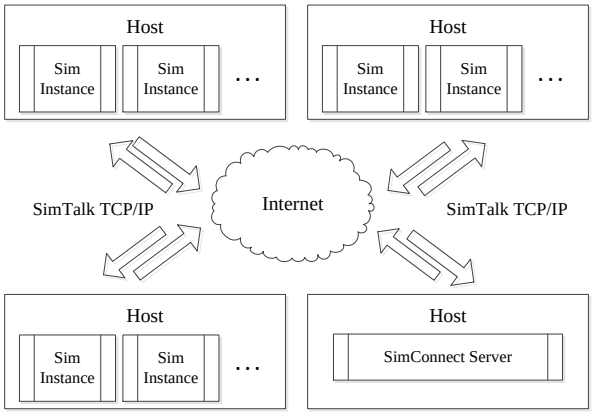


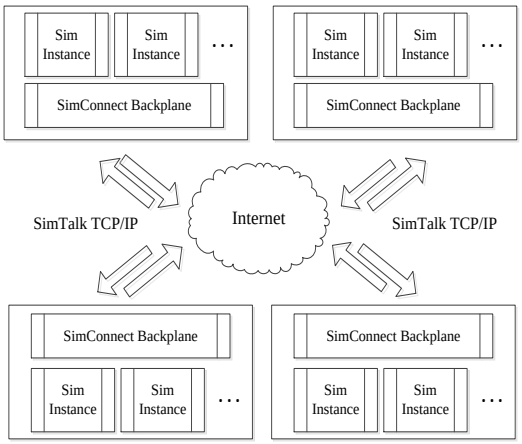
Illustration 18. Heterogeneous client-server hierarchy and network distribution

Illustration 18 shows how a CPS simulation of different component simulators maps to the SimConnect/SimTalk client-server architecture. Consider an example problem of source-level debugging of a real-time operating system (RTOS), controller area network (CAN) protocol stack, and application layer code over two virtual Freescale 9S12 microcontroller targets in the TExaS 9S12 simulator [30]. These controllers could be exchanging sensor information or actuating different components of the physical system, with a high-speed, noise-resistive CAN data link to the sensors and actuators. Simulation with component realism is desired. Let the cycle-accurate "soft-IP" for the CAN controller peripherals be provided in Verilog or VDHL, and an Ngspice deck for the CAN transceivers and physical layer. The microcontrollers communicate with the CAN controllers through a memory-mapped register interface simulated with cycle-estimating microcontroller ISA simulators. The clients all exchange signals through the SimTalk protocol [10] to the backplane. There is no limitation on where these elements

(clients and server) reside, only that they support the SimTalk over TCP/IP. The system simulation may be hosted over the Internet "cloud," for example, a single site LAN, or within one multi-core machine (process separated), or over a parallel high performance machine (Illustration 19).

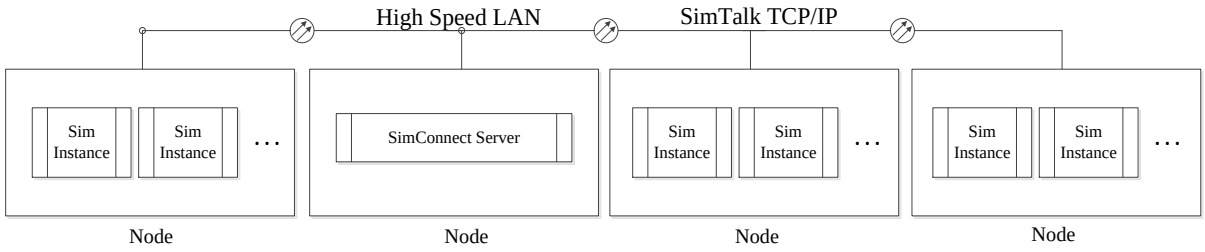


a. High Performance Cluster with single SimConnect backplane

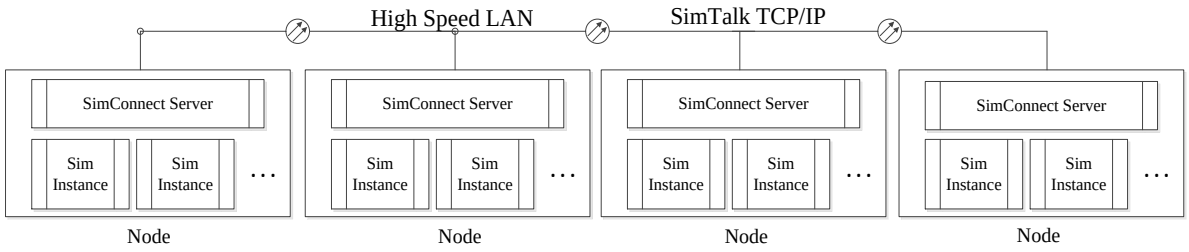


b. High Performance Cluster with single or multiple SimConnect backplanes

Illustration 19. Continued next page.



c. High Performance Cluster with single SimConnect backplane



d. High Performance Cluster with multiple SimConnect backplanes

Illustration 19. SimConnect/SimTalk client-server network distributions

To illustrate synchronization with an example, IEs can achieve conservative, predicted-event [16] synchronization, as employed in [10]. This synchronization example must coordinate the TExaS 9S12 microcontroller simulator, a time-driven, clocked synchronous model of computation (MoC), and Ngspice, a time-driven, dynamically stepped model of computation. IEs enable this. In the clocked synchronous MoC, input and output signal exchanges occur at ends of a fixed period, but a one evaluation cycle delay occurs from signal input to output result. For example, the TExaS simulator, a cycle-estimating simulator for the Freescale 9S12 microcontroller, continually executes a GetInputs(), Evalutate(), PostOutputs() cycle in its time advancement. With a local evaluation cycle of 125 ns, and static IE duration of 125 ns, there is a 125 ns delay from the operational effect of an input appearing on an output if they are related.

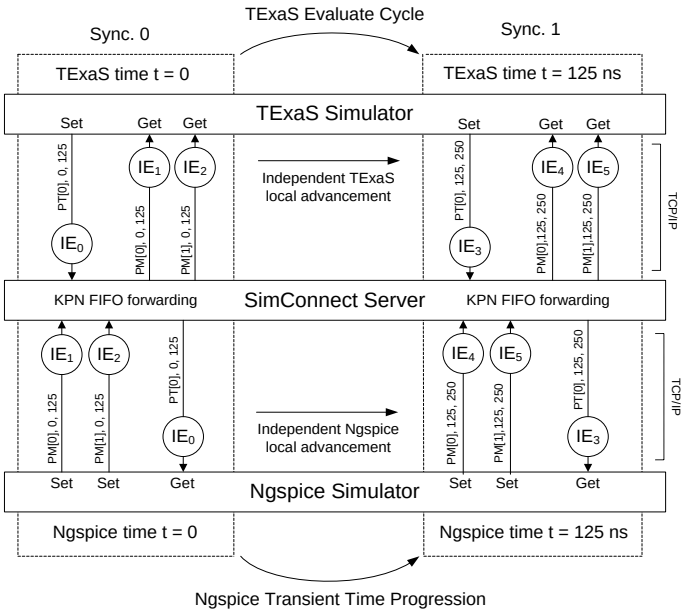


Illustration 20. Conservative, predicted event synchronization

In Illustration 20, the Xspice socket devices and TExaS post output IEs of $(t_n - t_m)$ = 125 ns duration, the predicted event interval, or the TExaS evaluation cycle. Initial condition IEs of 125 ns duration are posted to FIFOs at startup so each simulator can advance and post after the first Get() SimTalk operation. The IEs posted by TExaS, consumed by Ngspice, allow the Ngspice kernel to compute freely for 125 ns, posting IEs before blocking at sync point 1 in the illustration, where it re-queries the SimConnect server again. Up to each sync point, established by the expiration time of an input IE, simulators advance independently without local time coordination. For resolution, if the IE durations are less than the TExaS evaluation cycle, that is $(t_n - t_m) < 125$ ns, the FIFOs oversample. If the IEs are greater in duration, $(t_n - t_m) > 125$ ns, the execution is optimistic, since TExaS advances to the next evaluation cycle on an IE that could change during the evaluation cycle, but which was declared to be constant on an IE range larger than the cycle. If the IE duration increases further, $(t_n - t_m) >> 125$ ns, the signal

resolution decreases, decreasing message count, increasing the time between synchronizations, but decreasing accuracy. However, for low frequency input signals compared to the TExaS clock, resolution can be decreased to an appropriately large (t_n – t_m) period, say the period of the Nyquist frequency of the input signal. If an appropriate resolution is unknown, it can be observed first at a high resolution rate (t_n – t_m) << 125 ns, then adjusted to a lower resolution to increase simulation speed.

5.6 Software Metric Comparisons with HLA

SimConnect and SimTalk were designed to impose lightweight interfacing burdens on coordinated simulators from a software development and maintenance point of view. The SimConnect backplane was designed for implementation simplicity as a KPN token router compared to backplanes that tightly interface into coordinated simulator time management algorithms. Comparing software engineering efforts can be relative, but we can examine differences in code size for SimConnect and SimTalk versus some open source HLA RTI implementations and can look at an example of comparative application layer messaging for a time driven single simulation cycle in SC/ST compared to HLA. Because SimTalk may not require internal modification of proprietary simulator source code to interface to the simulation backplane SimConnect, the tools can result in reduced code complexity and potential messaging overhead compared to solutions that tightly interface simulator kernels.

5.6.1 Source Code Comparisons

Table 1 summarizes some of the software engineering metrics during SimConnect and SimTalk development.

Software Component	Man-months to Develop	Lines of Code	Files	Build Environment	Source Code Framework
SimConnect Server	2	1541	9	Linux 2.6.16, gcc 4.2.2	C source code
TExaS SimTalk connector	1	288	1	Microsoft Visual Studio 2010	TExaS C/C++ source code
Ngspice SimTalk analog port input connector	0.25	265	1	Linux 2.6.16, gcc 4.2.2	Ngspice user-defined device macros in C
Ngspice SimTalk analog port output connector	0.25	278	1	Linux 2.6.16, gcc 4.2.2	Ngspice user-defined device macros in C
Ngspice SimTalk digital port input connector	0.25	290	1	Linux 2.6.16, gcc 4.2.2	Ngspice user-defined device in C
Ngspice SimTalk digital port output connector	0.25	240	1	Linux 2.6.16, gcc 4.2.2	Ngspice user-defined device in C
Simulink SimTalk input port connector	0.25	250	1	Microsoft Visual Studio 2010	MATLAB .mex level-1 file
Simulink SimTalk input port connector	0.25	280	1	Microsoft Visual Studio 2010	MATLAB .mex level-1 file
Total	4.5	3432	16		

Table 1. SimConnect/SimTalk code sizes and development costs

In Table 1, the largest element of the implementation is the SimConnect backplane. The backplane is a stand-alone server written in the C language that implements dynamic IE FIFO management, socket connection management, SimTalk message parsing, dynamic resolution management, and IE token delivery. The SimTalk connectors for simulators TExaS, Ngsice, and Matlab implement SimTalk functionality in whatever code interfacing environment the simulator offers. The plugins provide socket management, SimTalk message parsing, and signal update and event scheduling through simulator APIs provided for user-specified devices. The SimTalk plugins require the ability to compile OS-level system calls, particularly to provide blocking reads on TCP/IP sockets, which enable the blocking IE token read functionality of the Interpolated Event Port specification of Chapter Three.

Software Component	Lines of Code	Files	Source Code Framework
CERTI 3.4.1 RTI [80]			
libCERTI/	28769	112	C++
libHLA/	7522	57	C++
libRTI/	12465	28	C++
Total	48756	197	
OpenHLA 1.3 RTI [82]			
hla-1.3/	3348	101	Java Classes
OpenHLA IEEE 1516 RTI [82]			
ieee-1516e/	3778	61	Java Classes
MatlabHLA13 [79]			
m_files/	2438	95	Matlab .m file format
rti.cpp	2466	1	C++
Total	4904	96	

Table 2. Open source HLA code metrics

There is no unilateral way to compare the SimConnect/SimTalk software complexity with implementations of HLA-based solutions [19], but Table 2 lists some source code metrics from open source implementations of the HLA: the French Aerospace Lab ONERA [81] CERTI suite [80], and the OpenHLA [82] implementation. Source code sizes for the HLA RTI backplane are much larger than the SimConnect backplane. This is because the HLA standard requires many other functions from the RTI besides time management and signal value communication. The RTI must manage simulator objects declared in the Federation Object Model (FOM) [19], manage simulator object update calls and interaction calls, calculate equivalent Lower Bound Time Stamps (LBTS) and Global Virtual Time (GVT) for conservative and optimistic coordination, and implement many other duties given in the standard [19][74]. The SimConnect backplane, however, must only implement the KPN dynamics, parse SimTalk messages, and route IEs to simulator IE port connections. This is the primary benefit of a dataflow

architecture compared to a library-of-classes: the functionality of simulator coordination in the dataflow architecture is shifted into the inherent dataflow dynamics as much as possible and removed from explicit APIs.

Based on differences in code size between the MatlabHLA13 solution [79] for interfacing Matlab/Simulink into an HLA federation, and the .MEX file code sizes for the Simulink SimTalk plugins of Table 1, we speculate that for a closed architecture commercial simulator, the software engineering effort of implementing a SimTalk plugin will hopefully be less than implementing an equivalent HLA ambassador.

5.6.2 Application Layer Messaging Comparisons

Messaging traffic comparisons can depend on message format and size, means of software transmission (such as BSD sockets, MPI, named FIFOs, or other solutions), means of packet transmission (such as TCP/IP), means of circuit level bus transmission (such as Ethernet or CAN or USB), means of physical transmission (copper wire, fiber optic, or wireless RF), and then environmental circumstances (EM noise, network demand, solar activity, and physical breakages). Therefore, comparison of messaging burdens between two solutions can require a very limited context of observation, but not hold true for all cases.

The HLA RTI specification [19] communicates with federate simulators through code layer classes called the RTI Ambassador and the Federate Ambassador (Illustration 22) [74]. The ambassador classes provide APIs for the RTI backplane to make requests upon a federate, such as a request to reflect object values that other federates have updated), and APIs for federates to make requests upon the RTI backplane, such as a request to advance time. An important difference between SimConnect/SimTalk and the HLA API structure is that *time and signal information are combined in one message in*

the KPN-IE method of SimConnect/SimTalk. That is, signal value and signal duration are combined into one IE formatted message.

If we take a restricted example where a SimConnect/SimTalk simulator is advancing a constant time step each cycle, and consuming one IE for *signal_a* and posting one IE for *signal_b* in *conservative mode*, the simulation application layer messaging cycle will appear as in Illustration 21.

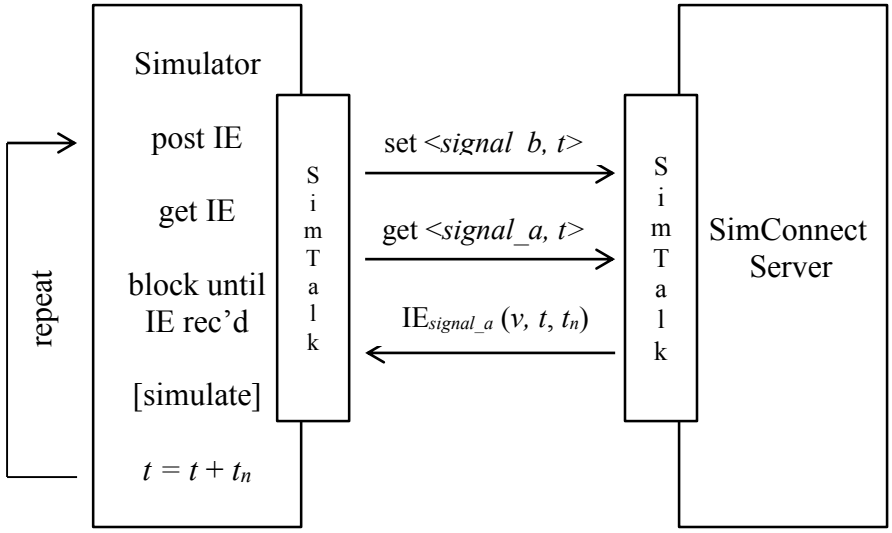


Illustration 21. SimConnect/SimTalk conservative mode single simulation cycle

At time *t*, the simulator will post an IE for *signal_b*, without blocking, and then issue a SimTalk get request to the SimConnect backplane for *signal_a*. The simulator will block until the backplane responds with an IE. When it does, the simulator is free to simulate and advance time to the IE t_n value, because *signal_a* is declared to be unchanging until t_n by the definition of an IE (Chapter 3.1). When the simulated time reaches t_n, the simulator updates *signal_b* with a post message and continues the cycle. In this mode, three messages per cycle occur between the backplane and simulator.

In the HLA case, in Illustration 22, there is some flexibility in the federate as to how it uses the various flavors of the time advance request functions, the reflect object

attributes function, and the update object attributes function. Illustration 22 assumes the federate is not using the "interactions" API offered by an RTI. Importantly, the RTI API separates messages of time management and signal information management. Because a time-constrained federate must surrender time advancement to the RTI until the RTI grants it a "Time Advance Grant" message [19][74] after a "Time Advance Request" message, there can be *two* messages between federate and RTI per simulation cycle in addition to signal updating messages (reflect and update object values). In the case of Illustration 22, which assumes a time-driven simulation cycle, with a time-constrained federate, and only one subscription object and one published object (no "interactions"), there can be four federate-RTI application layer messages per simulation cycle compared to the three of SimConnect/SimTalk Illustration 21.

This assumes that the conservative SimConnect/SimTalk simulator is not using the TxAck (Chapter 5.3) server to client message given in response to the SimTalk "set IE" message, which would add a fourth message per cycle. TxAck is required if the simulator wants to block on posted IEs until a destination simulator consumes them, or if the simulation has combined conservative and optimistic coordination, which can benefit from the TxAck message on posted IEs for LBTS and GVT bounds tracking (Chapter 4.2.2).

Also significant in the HLA RTI case is that federates and RTIs communicate through the ambassador code-level APIs, which interface into internal federate and RTI functionality. In the open source cases of Table 2, these ambassadors are implemented in C++ and Java classes, and the federate and RTI use them through class function call methods as the classes are compiled into the federate and RTI executives. This can make runtime debugging difficult, since the many APIs of the federate and RTI ambassador specification [19] can require connecting a debugger to a runtime executive to examine a

failure within a class method call or in the federate usage of an API. With SimTalk plugins, however, the only included libraries are those needed to leverage TCP/IP sockets. All other data exchange happens over the SimTalk messaging channel. Therefore, simulator signal and time debugging can occur externally to the simulator by tracking IE streams, rather than attaching a debugger to the simulator.

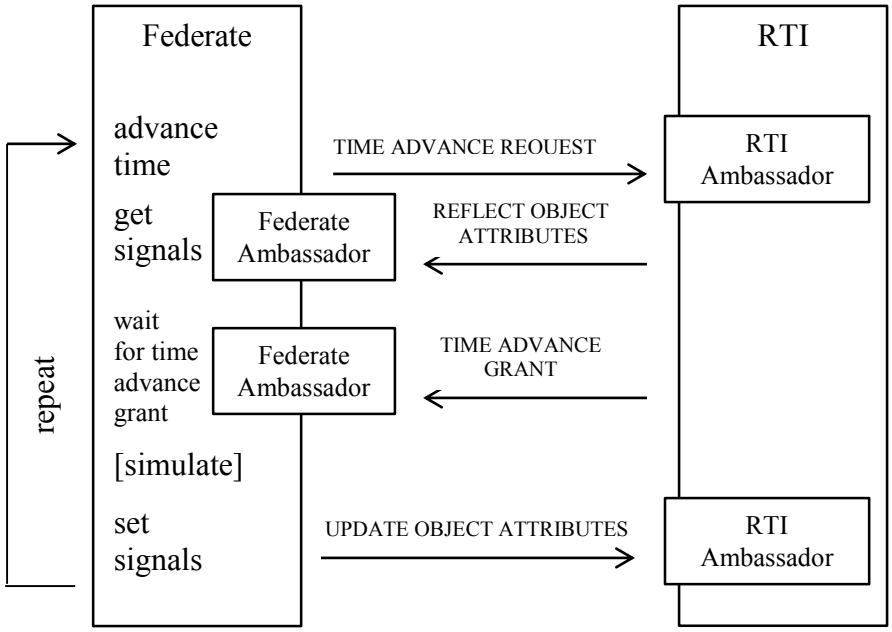


Illustration 22. Example conservative time driven federate-RTI simulation cycle [74]

5.7 Chapter Summary

The SimConnect and SimTalk tools implement the KPN-IE method for parallel and distributed simulation coordination. SimConnect, a simulator coordinating backplane, provides the behavioral and routing services of a Kahn Process Network, forwarding Interpolated Event tokens from connection producers to connection consumers in a publish/subscribe architecture. SimTalk, a messaging protocol, facilitates the exchange of IEs between a participating simulator and the SimConnect backplane.

Coordinated simulators connect to the SimConnect backplane through SimTalk in a client/server relationship and are decoupled from awareness of one another, communicating only with the SimConnect backplane. SimTalk is implemented over TCP/IP sockets to enable simulator distribution over networked computation resources. The ability of SimConnect/SimTalk to support dynamic simulation resolution by varying Interpolated Event resolution (ΔIE) was presented, highlighting the ability of any participant in the simulation to dynamically change resolution on a signal producer. An example of conservative, predicted events coordination between the TExaS simulator and Ngspice simulator was also presented. Finally, some differences between the SimConnect/SimTalk implementation of the KPN-IE method and open source HLA-based implementations were examined, revealing a smaller source code footprint for SimConnect/SimTalk. The significant functional differences required of the SimConnect KPN-IE backplane versus a compliant HLA 1.3 or IEEE 1516 [19] implementation may reduce the potential software engineering costs for integrating CPS simulators with the KPN-IE method compared to an HLA based solution.

CHAPTER SIX. EXPERIMENTS AND RESULTS

The SimConnect/SimTalk tools enabled the coordination of many homogenous simulators, leading to a novel result in Spice simulation acceleration [17], and the coordination of heterogeneous simulators [10][11] for realistic simulation of a cyber-physical control system. Results for the coordination tradeoffs in simulation resolution, speed, and accuracy are reported for each experiment, and results from dynamic simulation resolution are reported from [18]. Results show the scalability of SimConnect/SimTalk (up to 128 simulators) from [17], and the reach of SimConnect/SimTalk for coordinating three diverse simulators not previously coordinated with each other [11].

6.1 Homogeneous Coordination

6.1.1 DISTRIBUTED SPICE COORDINATION

In the case of homogeneous simulator coordination, SimConnect/SimTalk (SC/ST) was used to coordinate up to 128 distributed Ngspice simulators [17] for the parallel simulation of a counter circuit of over three thousand BSIM3 model transistors. Parallel speed up gains of up to 52x by software parallelism alone (Figure 2) were achieved with tunable tradeoffs in speed versus accuracy. This result in general can enable simulation acceleration by coordinating multiple instances of a simulator if the simulator offers a blocking, OS-level software interface, but has a closed internal architecture preventing study of internal parallelism.

Expression Level Parallelism

Spice based circuit simulation has long been an area of research for acceleration, due to the exponentially increasing simulation time of circuits as the number of circuit nodes increase. Parallelism can be a means to achieve Spice simulation speedup.

Expression-level parallelism starts at the model description layer. At this layer, the model is inspected for points of partition, at which nodes the circuit is expressed as new, independent subcircuits with communication interfaces. Each new subcircuit is assigned to an independent simulator. The entire model then simulates in coordination over the distributed, coordinated instances of the single simulator normally hosting the non-partitioned model. In this way, if a communication interface is offered at the model description layer, parallel execution may be gained for simulators not normally supporting internal parallelism. The cost for the approach is the additional communication overhead between simulators (both a computation and latency cost), and the burden of coordinating distributed simulators with independent versions of time advancement.

As an example of a partitioned distribution, Illustration 23 shows the coordination of eight Ngspice instances connected to the SimConnect server through SimTalk, for a concurrent 8x parallel simulation of the 128-bit counter described in Illustration 24. The counter is partitioned into subcircuits 16 bits wide, connected at their MSB and LSB nodes via Xspice [33] socket devices.

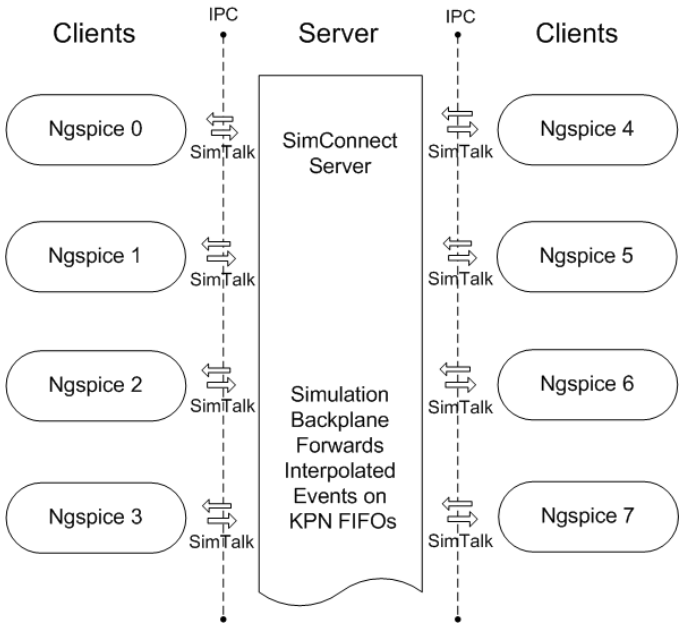


Illustration 23. SimConnect/SimTalk relationship for distributed, parallel Ngspice instances

Experiment Configuration

Consider simulating a wide-bit asynchronous ripple counter at the transistor level. While ripple counters are impractical as real circuits, due to the rollover delay from maximum value (0xFFF…) to zero, they are simple elementary circuits for conceptualizing or simulating a propagation delay (the rollover delay as the carry bit propagates from bit 0 to bit <n>-1, for counter width <n>). Consider the <n>-bit ripple counter in Illustration 24, composed of inverters and positive edge-triggered D flip-flops.

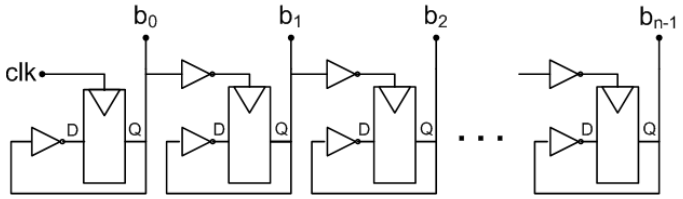


Illustration 24. <n>-bit asynchronous ripple counter

The inverters are implemented as a standard pmos/nmos pair, and the D flip-flop at the gate level is implemented according to Illustration 25.

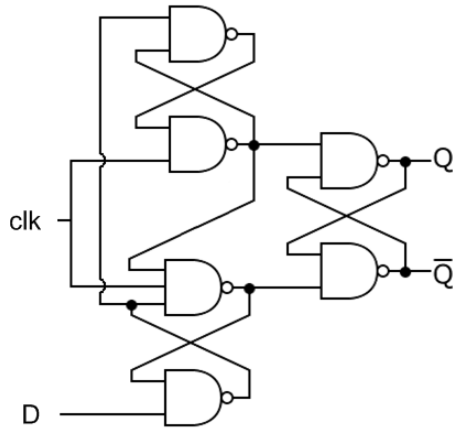


Illustration 25. Edge-triggered D flip-flop

Each bit, with two inverters and one flip-flop, consumes 30 MOSFETs, 15 pmos transistors and 15 nmos transistors. The pmos transistors are oversized for symmetric drive strength with respect to the nmos transistors.

Sequential Simulation

The single-instance counter is simulated in Ngspice [31], the open source distribution of Berkeley Spice version 3 and Georgia Tech's Xspice [33]. Figure 1 shows the increase in transient analysis time as the number of transistors in the circuit increases, per bit width of the counter. The counter is simulated at 4, 8, 16, 32, 64, and 128 bits for 1.5 μs of simulation time.

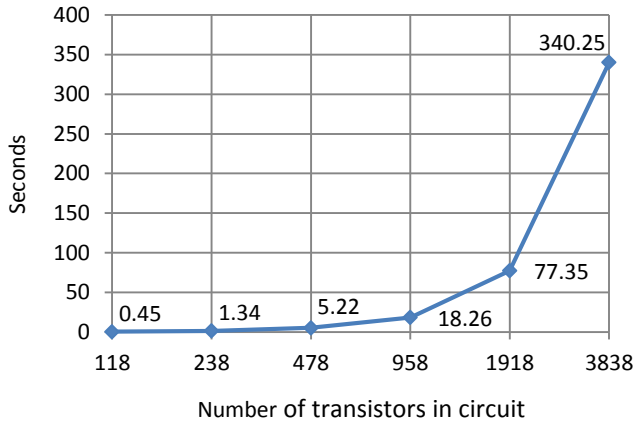


Figure 1. Ngspice transient analysis time for 1.5 µs of simulation time as counter width increases

From Figure 1, the increase in analysis time per number of transistors is non-linear due to the non-linear increase in model evaluation time and matrix solution time in the Spice kernel as the device count increases. As the number of bits in the counter increases from 64 to 128 bits (1918 to 3838 transistors), for example, the analysis time increases from slightly over a minute to more than five minutes on a single workstation Linux 2.6.16 kernel machine with an Intel Xeon 2.93 GHz processor. This non-linear increase limits the practicality of simulating complex circuits at the transistor level on the order of modern VLSI transistor counts.

Parallel Simulation

For improvement, the circuit is partitioned at the expression level (the Ngspice circuit deck) into subcircuits <m>-bits wide, where <m> is a power-of-two divisor of 128, and the factor of parallelization. Each subcircuit is then assigned to an independent Ngspice process, coordinated with other Ngspice processes in parallel through SimConnect and SimTalk.

Illustration 26 shows an <m>-bit wide subcircuit of the counter, where Xspice user TCP/IP socket devices connect the circuit to its neighboring subcircuits over SimTalk.

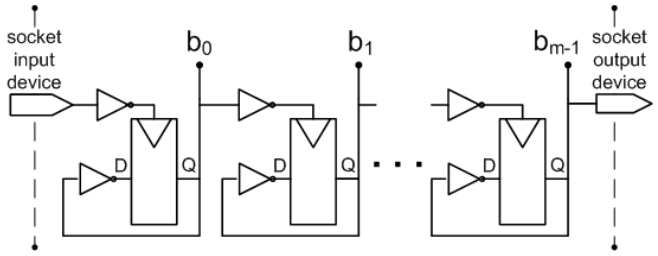


Illustration 26. Partitioned subcircuit with socket devices

Between subcircuits, TCP/IP socket devices connect the most significant to least significant bits from one subcircuit to the next. For example, bit 15 of the subcircuit for bits [0:15] is connected to bit 16 of the subcircuit for bits [16:31], and onward through bit 127. The socket device services the SimTalk protocol and delivers IE tokens to the SimConnect backplane, which distributes the IE tokens through KPN FIFOs from signal producer to signal consumer.

Speedup Versus Accuracy Results

At 10 ns IE resolution, Figure 2 shows the speedup result per factor of parallelization for the same 128-bit counter for 1.5 µs of transient analysis time.

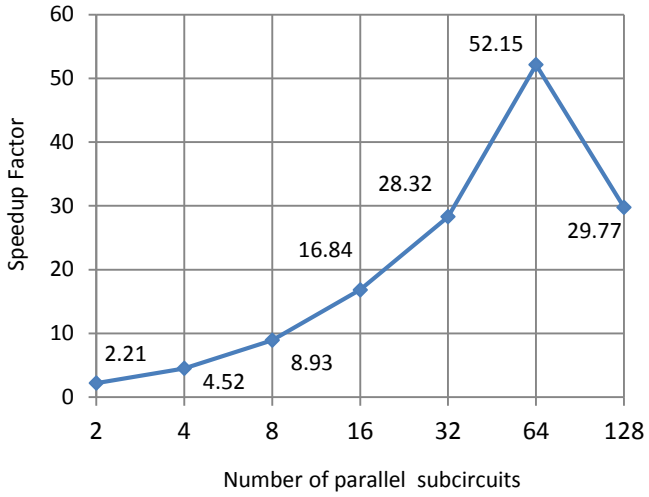


Figure 2. Speedup at 10 ns IE resolution

By dividing the 128-bit counter into two 64-bit subcircuits, we achieve a 2x speedup alone, and then achieve a 52x speedup by subdividing into 64 subcircuits, each 2 bits wide. However, the speedup maximizes at this point, after which it diminishes as the communication overhead per number of Ngspice instances increases. This manifests in the loss of speedup from 64x to 128x parallel in Figure 2. The cost of fixed-resolution IE duration also results in a non-zero percent error of measurement, shown in Figure 3, where the rollover time of the ripple counter across the parallel cases is measured against the rollover time of the non-parallel case.

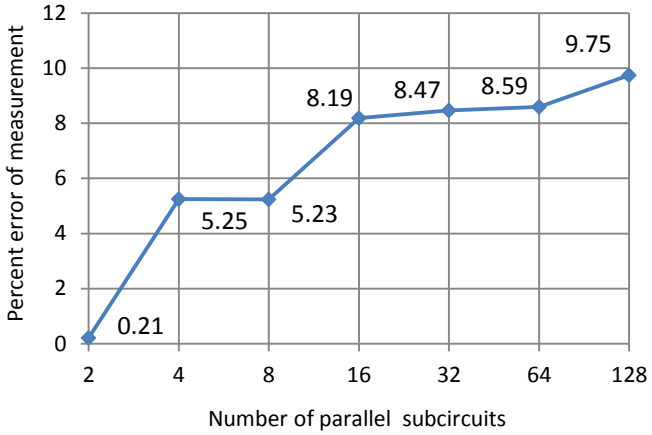


Figure 3. Percent error of measurement at 10 ns IE resolution

The error in measurement occurs because the IEs are of finite duration, during which sample time an event is declared constant. If an IE duration is greater than the rail-to-rail fall time of a circuit inverter, for example, or on the order of it, conveying the information of the inverter's changed state may be delayed up to the duration of the IE, depending on when the inverter output was sampled. Since this delay can continue from one communication node to the next through each parallel instance, it can accumulate at the output at bit 127 where the rollover delay is measured. The sum of accumulated delay can increase as the parallelism increases. This is responsible for the positively correlated relationship in Figure 3.

Increased Resolution

However, if IE resolution increases (from 10 ns to 2 ns) in Figure 4, the percent error of measurement decreases. This is because an inverter fall at communication nodes is sampled every 2 ns, instead of 10 ns. Since the inverter fall time is on the order of 10 ns as these transistors were sized, a 2 ns sample results in smaller worst-case delay in observing a rail-to-rail state change on an inverter output. Percent error of measurement

drops to below five percent for the 64x and 128x parallel cases in Figure 4, and to below one percent for the 2x to 16x parallel cases, although the speedup decreases.

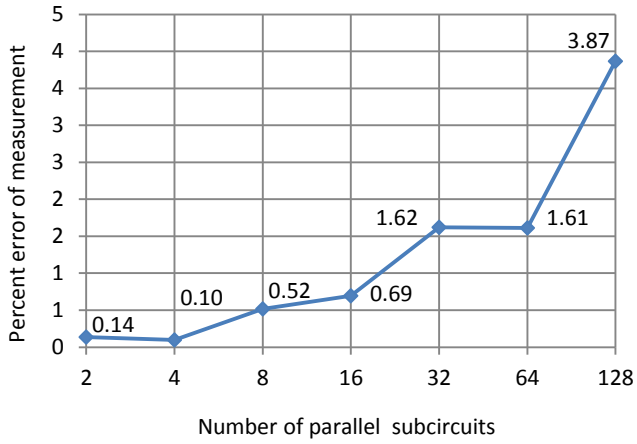


Figure 4. Percent error of measurement at 2 ns IE resolution

The transient analysis time for the same degree of parallelism increases as resolution increases, shown in Figure 5, due to the increased communication rate with the SimConnect server. The higher resolution results in smaller Ngspice time steps, resulting in more IE tokens through the KPN FIFOs.

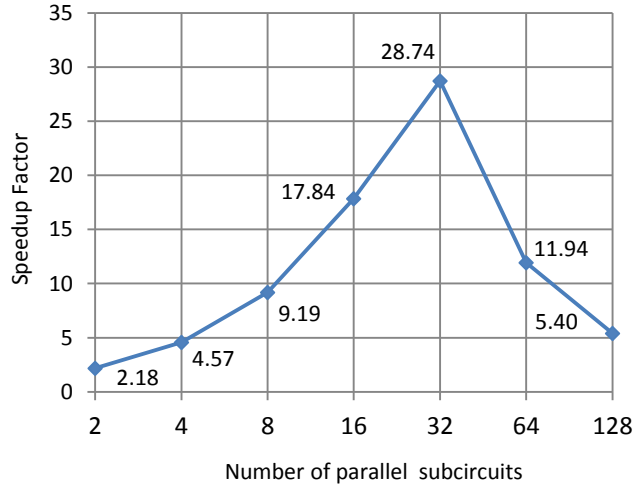


Figure 5. Speedup at 2 ns IE resolution

6.1.2 DISTRIBUTED SPICE SUMMARY AND CONCLUSIONS

Significant transient analysis time decrease (17x at less than 1 percent error) may be achieved by partitioning the 128-bit counter into subcircuits each 128/<n> bits wide, where <n> is a divisor of 128 and the desired factor of parallelism. Maximum speedup occurs at a parallelization factor per IE resolution, beyond which, as parallelization increases, speedup decreases due to increased communication and load on the SimConnect backplane.

This cost of communication also occurs as IE resolution increases. However, percent error of measurement can be reduced arbitrarily per degree of parallelization by increasing the IE resolution, as shown in Figure 5. This speedup in transient analysis time for the same 128-bit counter is achieved without modifications to the Ngspice kernel, or the execution host, making it different than execution-level parallelization schemes. In this method, the partitioning is performed at the circuit expression-level, in multiple Ngspice decks spread over independent simulators, so both the model-evaluation and the matrix-solving phases occur in parallel.

Choosing an appropriate degree of parallelization and IE resolution automatically is not yet suggested by this work, since it is highly circuit dependent (the partitioning should look for nodes of loose coupling or signal feed-forward cutsets). For accuracy, IE resolution should be on the order of the maximum frequency content of the communicated signal to minimize accumulated delay of rise or fall time information due to sampling. In the example of Figure 5, decreasing the IE duration to one fifth (2 ns) of the circuit inverter rail-to-rail fall time (approximately 10 ns) decreased the percent error of measurement for each degree of parallelism by more than one-half for the 4x through 128x parallel cases.

With this approach, there will always be tradeoffs between degree of parallelization, total circuit analysis time, IE resolution, and percent error of measurement. However, gains up to 52x transient analysis time at less than ten percent error by this software technique alone, without modifying the simulator or execution host, may be acceptable at some early investigation phases of system-level design (SLD) [3].

It may be possible to combine expression-level parallelization and execution-level parallelization for further speedup. For example, if at execution-level, a K-times speedup is achieved, then that same speedup would be achieved individually over <N> separate Ngspice instances, since the speedup is internal to each instance. If at expression-level, though, a J-times speedup is achieved, then combining both, a J times K factor of speedup should be achieved for both techniques (the speedups should multiply, not add). One speedup occurs at the execution-level, another at the expression-level. There will still be an error in measurement due to the usage of IEs with this method at the expression-level (compared to execution-level methods that may or may not introduce error). It may be possible to apply this technique to simulators not initially written for parallel internal execution, such that they offer a communication interface capable of hosting a SimTalk connector, to see if similar speedup gains can be achieved.

6.2 Heterogeneous Coordination

6.2.1 Distributed PID/PWM Software-based Motor Control

SimConnect and SimTalk (SC/ST) were applied to simulate a software-based PID/PWM digital controller of a DC motor with electrical, instruction set-level, and physical model realism [11]. This is significant for demonstrating the modeling range of SC/ST based simulations as more SimTalk connectors are written for new simulators. The many world-physical effects that can be simulated in multiple, coordinated

Matlab/Simulink instances are enabled by the technique. In Figures 10 and 11, the PID/PWM controlled run up time of the 2nd order, 6-parameter DC motor model is plotted across different simulator and IE configurations to verify accuracy. We progressively build up the model in terms of realism and heterogeneity.

Experiment: 1-Simulator classical continuous PID controller and second-order DC motor model in Simulink

For a truth condition, we model the DC motor initially in Simulink in continuous time as a 2nd order system in the Laplace domain, with a transfer function given in Illustration 27. The model is taken from [37], where it is derived from first principles of KVL, KCL, and Newton's laws applied to rotation. The transfer function in the complex frequency domain *s*, where *V* is the applied terminal voltage in Volts and Θ is the rotor output position in radians.

$$\frac{\Theta(s)}{V(s)} = \frac{K}{s(\tau(s)+1)}$$

$$\text{s.t. } K = \frac{K_t}{bR+K_tK_e}$$

$$\text{and } \tau = \frac{RJ}{bR+K_tK_e}$$

Illustration 27. 2nd order DC motor model transfer function [18]

The model is parameterized for the simulation as follows:

R	motor terminal electrical resistance	1.0 Ohm
L	motor terminal inductance	0.001 Henry
K_t	torque constant	0.1 Nm/Amp
K_e	electrical constant	0.1 Nm/Amp
b	rotor viscous friction coefficient	0.001 Nm·s
J	rotor moment of inertia	0.01 $kg \cdot (\frac{m}{s})^2$

Table 3. DC motor model parameters

The torque constant K_t represents the electro-mechanical multiplier of the armature current to rotor torque. The electrical constant K_e represents the multiplier of back electromotive force (back-EMF) to rotor speed. The electrical equivalent circuit is given in Illustration 36. Coefficient b is a drag force, and coefficients L and J are integrating resistances to applied voltage and applied torque which go to zero in the steady-state with a constant terminal voltage. The motor runs up to a steady state speed as the back-EMF increases per the rotor speed, and current equalizes to meet resistive and friction losses.

The model is converted to a Simulink block-diagram form as summing, integrating, and gain blocks in Illustration 28, illustrating the feedback relationship between the motor electrical and mechanical dynamics.

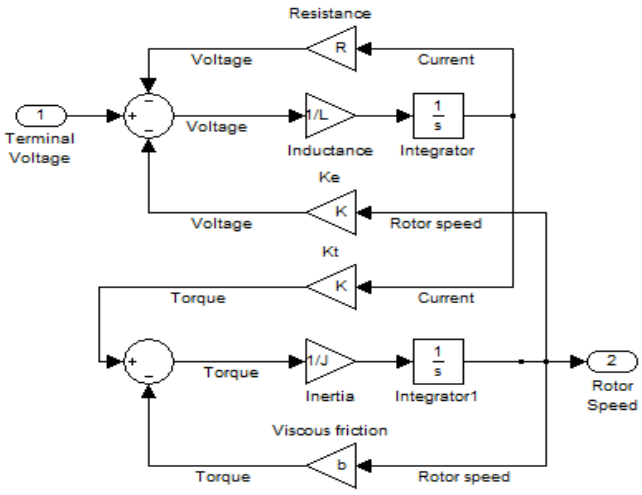


Illustration 28. Simulink DC motor electro-mechanical model

The model is encapsulated as a Simulink subcircuit, and an open-loop 5 Volt step-function is applied in Illustration 29 to achieve the no-load, open-loop speed run up plot in Figure 6.

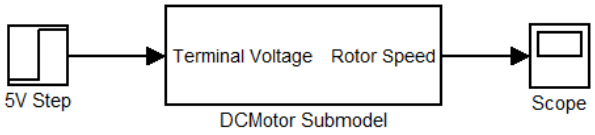


Illustration 29. Simulink open-loop 5V step-function stimulus

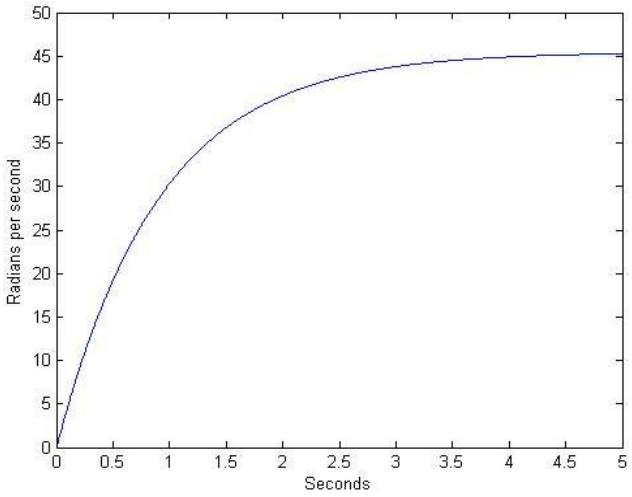


Figure 6. Model rotor speed versus time, open-loop transient response to a 5 Volt step function

Result: 1-Simulator classical continuous PID controller and second-order DC motor model in Simulink

With the no-load 50 radians/s speed as a ceiling, we add a Simulink continuous time PID block in Illustration 30, configured to a set point of half-speed 24 radians/s, 5 Volt output ceiling, with K_p, K_i, and K_d coefficients of 8, 2, and 1 respectively. The closed-loop transient response is plotted in Figure 7, with controller effort from the PID block in Figure 8.

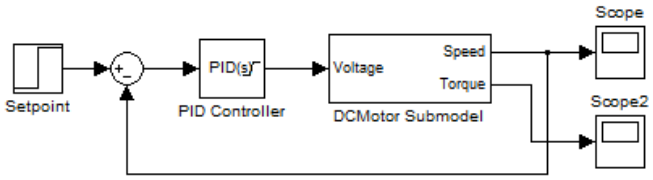


Illustration 30. Simulink continuous PID controller

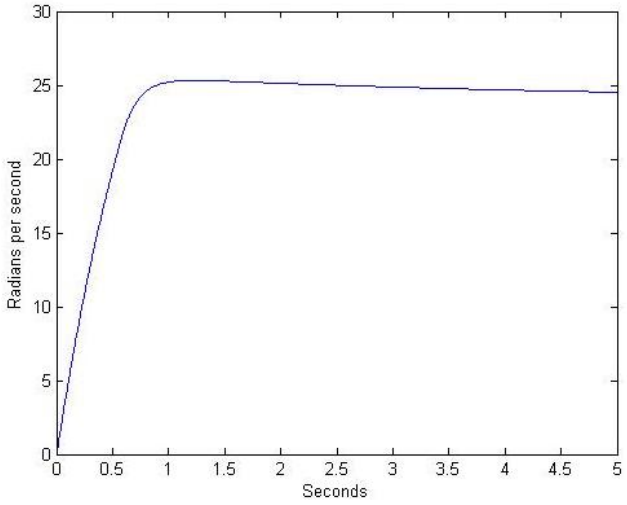


Figure 7. Model rotor speed versus time in Simulink continuous PID controller closed-loop transient response

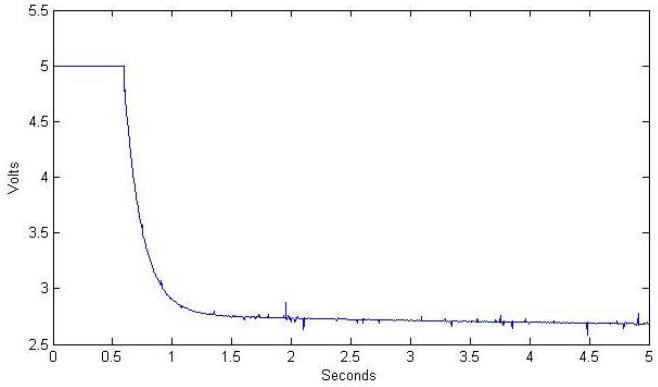


Figure 8. Model controller effort in Simulink continuous PID controller transient applied voltage

There is an expected overshoot in the continuous controller model with the given coefficients and lack of any limiter such as anti-integrator windup [30]. From the controller effort, the PID block outputs full ceiling power (5 Volts) until the set-point is approached, after which it drops to the steady state output necessary to equalize electrically resistive and mechanically viscous friction losses at the constant speed.

For a first departure away from the idealized continuous model, we quantize the motor speed output to a range of 128 values with a Simulink 8-bit quantizing block with an offset given in Illustration 31. This allows us to express the set point as one-half (0x40) of full value (0x7F) rather than an absolute rotor speed, and serves as an abstracted 7-bit analog-to-digital converter (ADC) that will be used as in input to the software-based controller. The effect on controller effort from quantizing the measured speed for the PID transient response is given in Figure 9.

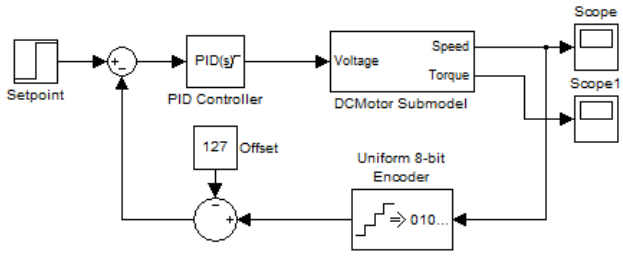


Illustration 31. Simulink quantized PID controller

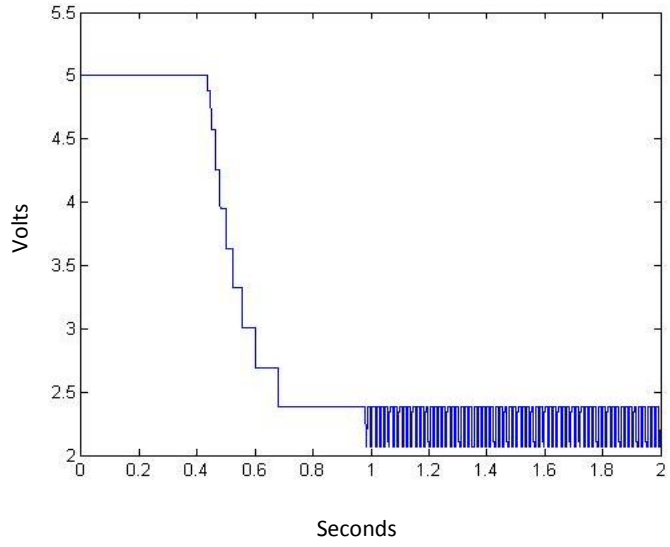


Figure 9. Quantized output controller effort in Simulink quantized PID controller applied voltage

Experiment: 2-Simulator Digital Software PID with PWM actuator in TExaS and Simulink DC motor model

Next, the controller is refined to a software-based PID difference-equation algorithm with PWM actuators hosted on the 9S12 microcontroller, simulated in the TExaS simulator at the cycle-estimating, instruction-set architecture (ISA) level. The co-simulation signal structure is given in Illustration 32.

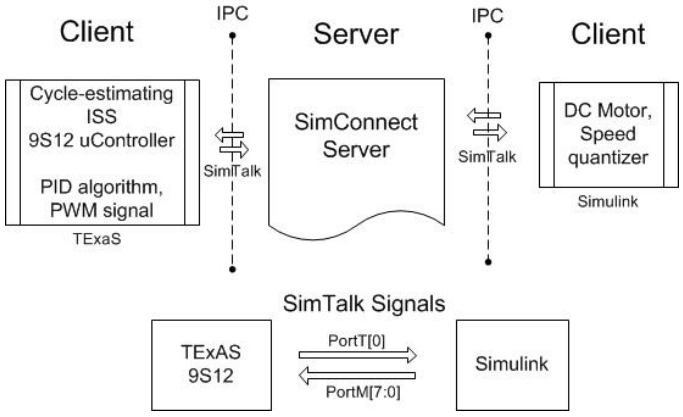


Illustration 32. 2-Simulator configuration

The software-based PID algorithm in 9S12 assembly is adapted from [30]. The PID coefficients are changed to K_p=24, K_i=2, and K_d=1 for the difference equation PID coefficients in fixed-point arithmetic. Conversion of a continuous-time frequency-domain specified controller to a digital controller is covered in [37]. Setup and allocation code for the software out of 9S12 reset and PID assembly is omitted for space, but is given in [30]. This implementation uses a free-running 1 kHz sampling rate PID main loop of 63 9S12 assembly instructions, and a total code length of 158 instructions. The algorithm also incorporates anti-integrator windup and output limit checking. The refined Simulink model is given in Illustration 33. The "socket_input" and "socket_output" S-Functions register SimTalk signals PortT[0] and PortM[7:0] for exchange with the SimConnect server. The PortT[0] digital signal is the PWM wave generated by TExaS. The 0/1 signal is amplified to 5 Volts for application to the motor terminals. The PWM wave and voltage in this configuration is modeled as ideal (zero rise/fall time).

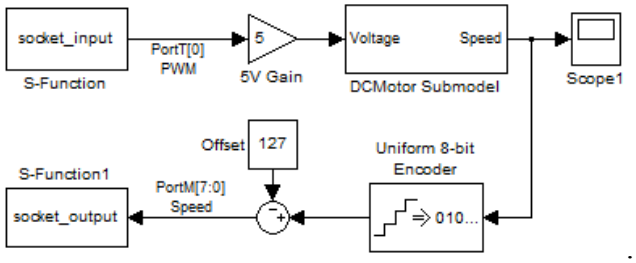


Illustration 33. Simulink DC motor model with SimTalk I/O interface

Results: 2-Simulator Digital Software PID with PWM actuator in TExaS and Simulink DC motor model

The 2-simulator model is conducted at 100 µs IE resolution on signals PortT[0] and PortM[7:0]. The transient response is plotted against the Simulink-only classical

continuous and encoded continuous cases in Figure 10. The times and traffic rates of the simulation are given in Table 4.

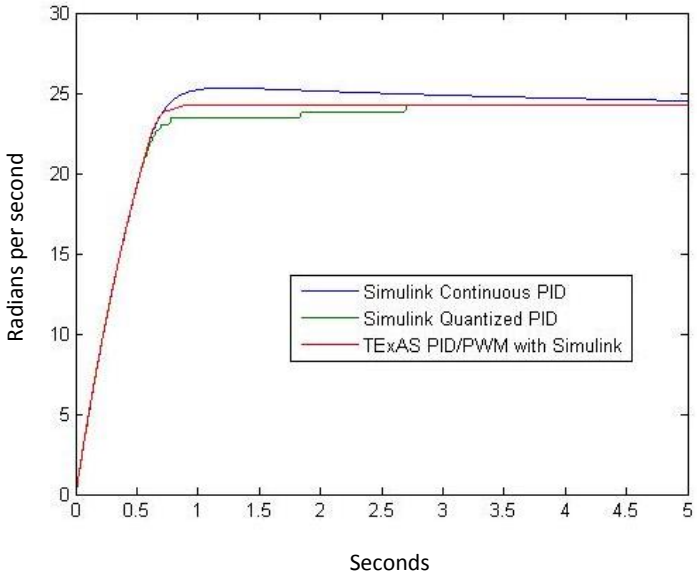


Figure 10. Model output speed versus time with Simulink-only and 2-simulator PID control model cases

The functionality of the software-based PID controller is verified in Figure 10, as the set point is reached in the steady-state. The set point approach differs from the classical case due to sampling, digitization, increased proportional gain, and anti-integrator windup. However, there is new departure from the continuous model because the applied terminal signal is a PWM signal from the 9S12 microcontroller, and the PID algorithm is realized in the microcontroller software. The 100 μs 2-simulator response is used as a baseline for checking the 3-simulator case, where electrical realism in the motor driver is added to the simulation.

Experiment: 3-Simulator Digital Software PID with PWM actuator in TExaS, electrical driver and DC motor model in Ngspice, and Simulink DC motor mechanical model

In the final configuration, electrical realism is added by modeling the motor driver circuitry and motor electro-mechanical model in Ngspice, duplicating the motor mechanical model in Simulink for output speed. Illustration 34 shows the cosimulation structure.

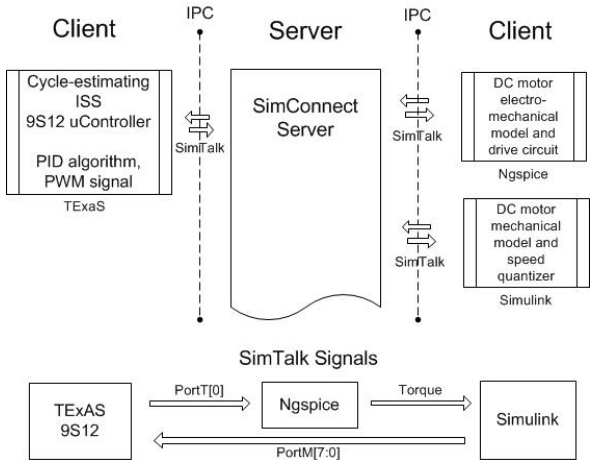


Illustration 34. 3-Simulator configuration

The PWM driver circuit is adapted from [30] and given in Figure 17. Sometimes called a "chopper" circuit [38], the power MOSFET circuit in Illustration 35 takes the low-power PWM PortT[0] PWM signal from the 9S12 and amplifies it across the motor terminals to the power voltage. When the PWM signal is high (5 Volts), the MOSFET is fully on, so current flows through the motor coil, and when the PWM wave is low (0 Volts), the MOSFET is off, interrupting the flow of current from the power source. There is still current flow when the MOSFET is off, however, due to the back-EMF and impedance of the DC motor. The high-voltage back-EMF and impedance when the

PWM wave changes, is collected through the 1N4004 “flyback” diode to protect over-voltage at the MOSFET drain.

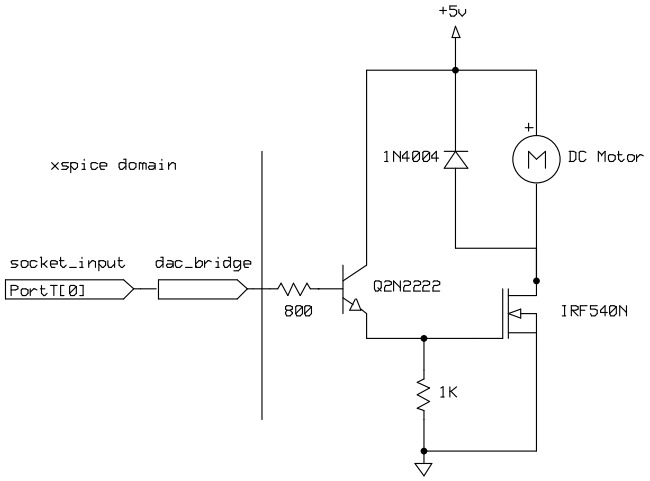


Illustration 35. Ngspice model for motor driver circuit

The DC motor electro-mechanical equivalent circuit is modeled in Illustration 36, where the applied torque is captured with the Xspice SimTalk “socket_output” device for delivery to the Simulink model.

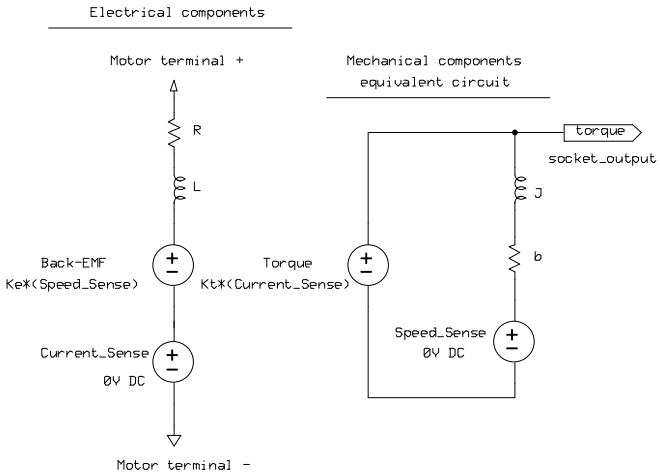


Illustration 36. Ngspice models for DC motor electrical and mechanical components

The sources “Back-EMF” and “Torque” are Ngspice Current-Controlled Voltage Sources (CCVS) driven from the current sensed in 0V DC sources “Current_Sense” and “Speed_Sense.” The mechanical components are modeled by their model-equivalent electrical components: an inductance for the rotor inertia, resistance for the viscous drag force, and voltage source for the torque. The Ngspice deck for the DC motor subcircuit and driver is given in Illustration 37, with parameters from Table 3 as in the Simulink model. Spice-3 compliant subcircuit models for the IRF540N, 1N4004, and Q2N2222 semiconductors are downloaded from [83] and [84].

```
V_Power 1 0 DC +5V

*Electrical motor components
D0 5 1 1n4004rl
R0 1 1 1
L0 2 3 .001
H_Back-EMF 3 4 v_Speed_Sense .1
V_Current_Sense1 4 5 dc 0V

* Mechanical motor components
H_Torque 10 0 V_Current_Sense .1
LJ 10 15 .01
Rb 15 16 .001
V_Speed_Sense 16 0 dc 0

* Power MOSFET
R1_pulldown 20 0 1000
X_IRF 5 20 0 irf540n

* IRF540N gate driver
R2 20 21 800
Q2 1 21 22  Q2N2222
```

Illustration 37. Ngspice deck for motor and driver

Two SimTalk sockets in the Ngspice deck provide the co-simulation I/O, an input to capture the 9S12 PortT[0] PWM wave output, and one output sample the circuit motor torque for consumption by Simulink. The Xspice and Simtalk components are specified in Illustration 38.

```
* SimTalk Inputs *
*
a4 100 socket_input_a
.model socket_input_a d_socket_input
+          (signal_name="portt0"
+           ip_address="SimConnectServer" port=8000)
*
a2 [100] [20] dac_bridge0
.model dac_bridge0 dac_bridge (out_low=0.0 out_high=5
+          out_undef=0
+           input_load=1.0e-12
+           t_rise=1.0e-8 t_fall=1.0e-8)

* SimTalk Outputs *
*
a5 [10] socket_output_a
.model socket_output_a a_socket_output
+          (signal_name="motor_torque"
+           ip_address="SimConnectServer" port=8000)
+          update_period=.00001
+           initial_value=0
+           initial_duration=.00001)
```

Illustration 38. Ngspice SimTalk devices 100 µs IE resolution

Finally, the Simulink model for the DC motor replicates the mechanical components for speed sensing in Illustration 39, and reports back to TExaS through SimTalk signal "PortM[7:0]."

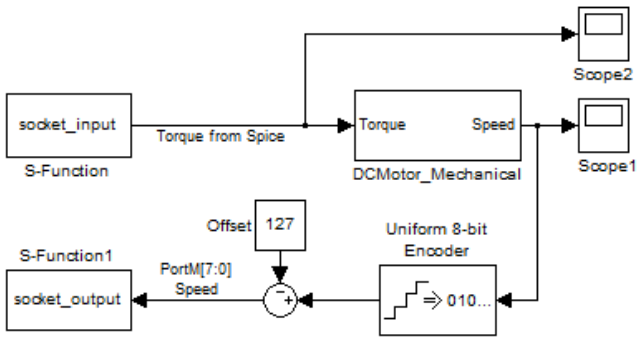


Illustration 39. Simulink co-simulation model with mechanical only DC motor model

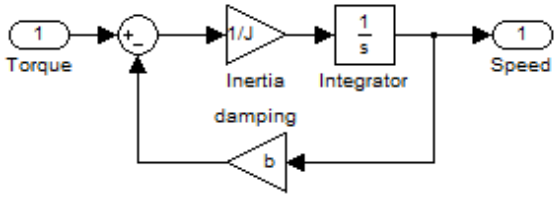


Illustration 40. Simulink mechanical only DC motor submodel

Results: 3-Simulator Digital Software PID with PWM actuator in TExaS, electrical driver and DC motor model in Ngspice, and Simulink DC motor mechanical model

The 3-Simulator configuration was executed over IE resolutions 10 µs, 50 µs, and 100 µs to measure speed versus accuracy against the 2-simulator case at 100 µs IE resolution. Speed of the 3-simulator execution is affected by the IE event rate (SimConnect traffic and time points) and internal simulator rates. In Figure 11, the 3-simulator case rise time is plotted against the 2-simulator case baseline.

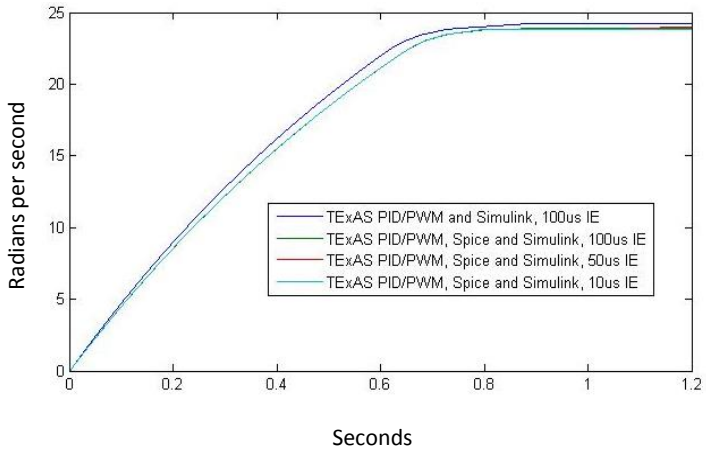


Figure 11. Model speed versus time in 2 and 3-Simulator configurations

As can be seen in Figure 11, the motor speed output profile as the electrical driver realism is added in Spice in the 3-Simulator case agrees with the 2-simulator case output to within ten percent error of measurement. The difference in output speed at a time point

in Figure 11 for the 3-simulator cases against the 2-simulator cases is due to the realism added in the electrical driver, where a voltage divider is created between the motor coil R and the IRF540N MOSFET R_{on} resistance, to a voltage divider ratio of 1/1.077. This results in the motor coil not seeing a full 5V power when the MOSFET is on, but 1/1.077 less, where in the 2-simulator case, the electrical driver is ideal and created through a Simulink 5 V gain block. Significant in Figure 11 is that the motor model profile agrees when modeled in two completely different simulators (Ngspice and Simulink), and the PID controlled speed output agrees in regard to rise time and steady state. Figure 11 also indicates that from 100 µs to 10 µs IE resolution, there is not a significant difference in output profile.

In Figure 12, however, as the IE resolution is decreased for simulation speed, the measured rotor speed output begins to depart from the baseline result.

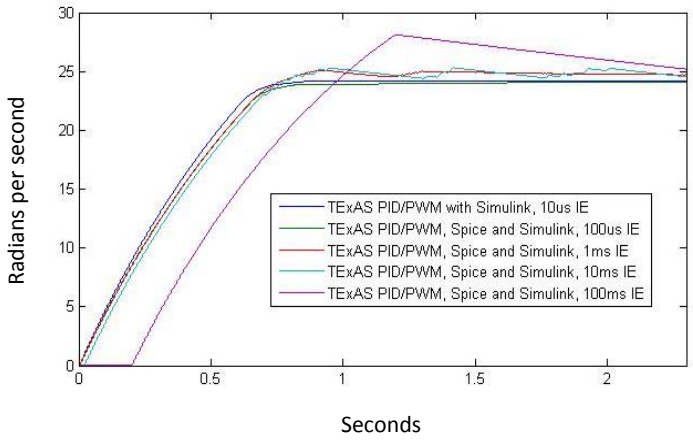


Figure 12. Variation in model rotor output speed versus time as a function of IE resolution

The departure from the control case in Figure 12 can be attributed to the coarse resolution in the IE period of measuring signals varying at 1 kHz in the PortT[0] PWM

wave signal. In the 100 ms IE case, for example, the IE is not able to pick up the transition of the PWM wave to high until 100 ms into the simulation as the initial condition IE is set to zero (and expires at 1 ms). If at 100 ms the signal happens to be sampled at zero, the consuming simulator (Ngspice) will process that value until its next expiration time at 100 ms later. This results in the Ngspice motor model not getting a power value in Figure 12 in the 100 ms case until 250 ms into the simulation. As the PID algorithm samples at 1 kHz, its PortM speed input is only reported every 100 ms, resulting in not recalculating a new speed value until every 100 PID loops, and any PWM updates only being sampled every 100 ms. As a result, although the simulation runs faster, the accuracy of measured output begins to decrease.

Speed versus accuracy in the 3-simulator case

Figures 11 and 12 indicate that the IE resolution for a cosimulation should be scrutinized against the bandwidth of the signals sampled by the IEs. The 100 ms IE resolution in Figure 12 curve five does not meet the period of the PWM wave in the simulation at 1 ms. However, a ceiling of 100 μs in the simulation meets the baseline of the 2-simulator case, and increasing the IE resolution does not appreciably change the accuracy of the observed rotor output speed.

Another point of comparison is to look at the controller effort in the 3-simulator case against the truth condition in the 1-simulator classical PID case. This is a signal with more variation over time than the controller rotor speed. The plot of the applied motor torque in two cases is given in Figure 13.

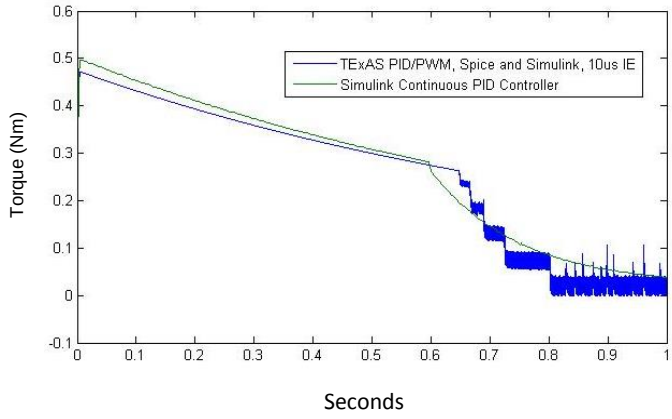


Figure 13. Discrete versus continuous model controller effort, applied motor torque versus time in 1-simulator and 3-simulator cases

The applied motor torque is the motor current through the armature times the motor K_t coefficient (0.1). With a coil resistance of 1 Ohm, this plot also tracks the curve of the applied terminal current. The difference between the applied torque through 0.5 seconds in Figure 13 is due to the electrical realism in the 3-simulator case of the motor coil resistance and IRF540N on resistance voltage divider, reducing the applied torque by a factor of 1/1.077. Through 0.5 seconds the PWM wave is 100% in the discrete case, and at maximum value (5 Volts) in the continuous case. The step nature of the discrete output is due to the quantized PID speed and finite sample rate of the PID algorithm in software.

In Figure 14, the controller effort is plotted against the continuous case for 100 µs and 10 µs IE resolution and zoomed to 40 ms. The offset from the continuous case is again due to the voltage divider electrical realism also seen in Figure 12. The plot, however, shows the significance of an IE resolution matching the bandwidth of changing signals shared between simulators. The spread effect in the 100 µs case is due to the time constant of the RL motor circuit as the PWM wave is held constant over a 100 µs IE

versus a 10 μs IE. The resolution does not affect the general curve or the PID rotor output, but RLC transient effects monitored in the Ngspice circuit will be more extreme. Guidelines for choosing digital controller sample rates are given in [37], and it is suggested here that the same approach apply to IE resolutions for simulated control systems.

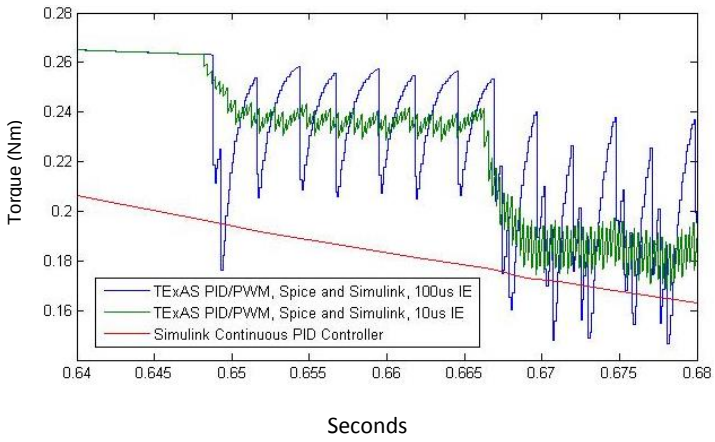


Figure 14. Discrete versus continuous model controller effort, applied motor torque versus time in 1-simulator and two 3-simulator cases

Execution times, counters and software factors

Table 4 summarizes the different execution times, configurations, and metrics for reported co-simulation configurations.

Column Legend

A Simulators: Simulink(S), Ngspice(N), TExaS(T)
B Interpolated Event (IE) resolution
C Number of SimTalk connectors
D Number of SimTalk messages (times 10^4)
E Number of 9S12 cycles (times 10^6)
F Number of 9S12 instructions (times 10^6, rounded 10^5)
G Simulated Time (seconds)
H Simulation Execution Time (minutes: seconds)
I Number of host machines

Data

A	B	C	D	E	F	G	H	I
S, T	10 µs	4	200	20	11.8	5.0	6.39	1
S, T	100 µs	4	20	20	11.8	5.0	3:07	1
S, N, T	10 µs	6	81.6	5.44	3.2	1.36	3:45	2
S, N, T	100 µs	6	12	8	4.7	2.0	3:36	2
S, N, T	1 ms	6	1.8	12	7.1	3.0	3:12	2
S, N, T	10 ms	6	0.3	20	11.8	5.0	4:09	2
S, N, T	100 ms	6	0.03	20	11.8	5.0	3:07	2

Table 4. Simulation times, configurations and message traffic

6.2.2 HETEROGENEOUS SIMULATION SUMMARY AND CONCLUSIONS

SimConnect and SimTalk enabled distributed, heterogeneous hardware/software co-simulation of three independent simulators, TExaS, Ngspice, and Simulink, modeling of a PID/PWM control system. The co-simulation and was tested against a baseline truth condition of a single Simulink simulation of the controller PID response and DC motor speed.

Significant to the SimConnect/SimTalk architecture is that once a SimTalk plugin is written for one simulator, it can communicate through the SimConnect server to any other simulator supporting a SimTalk plugin, for combinatorial growth in the number of simulator configurations possible. This can differ from 2-simulator or ad-hoc approaches written with specific simulator structures in mind. SimConnect/SimTalk meets a design requirement of source-based debugging with the ability to pause the global simulation by

breakpoints in the software-simulators by interrupting the KPN dataflow. In these experiments, when a breakpoint was inserted in TExaS, or the PID algorithm was single-stepped in the TExaS debugger, the Simulink and Ngspice interactive plots paused and advanced accordingly, without trace intrusion. This adds circuit-level inspection during the simulation as well as register-level inspection in software source debugging.

6.3 Dynamic Resolution in Heterogeneous Coordination

Continuing the 2-simulator model configuration of Chapter 6.2, the simulation is conducted first at a static 100 µs IE resolution on signals PortT[0] and PortM[7:0]. The transient response is plotted against the Simulink-only classical continuous and cases in Figure 15 to verify the functionality of the distributed modeling of the digital PID/PWM microcontroller-based simulation versus the Simulink continuous-time non-distributed controller.

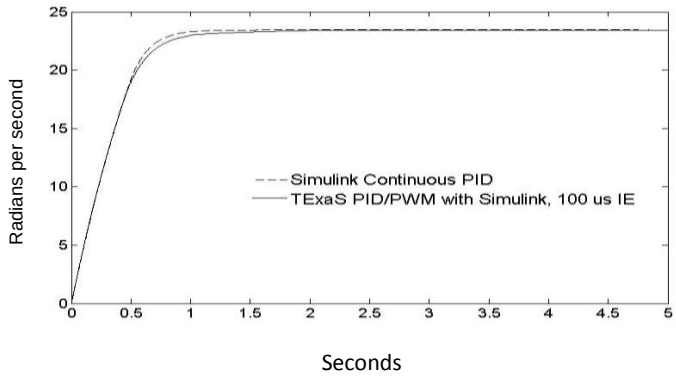


Figure 15. Model output speed versus time with Simulink-only and 2-simulator PID control model cases

In Figure 15, there is some departure from the continuous model because the applied terminal signal is a PWM wave from the 9S12 microcontroller in a software PID loop. However, the control profile shows set point agreement. The static 100 µs 2-simulator Case A is used as a baseline for checking dynamic resolution experiments, with

simulation times given in Table 5. For Case B, the static simulation localized to one machine to demonstrate the effect of network latency as a distribution cost.

In Case C, Figure 16, the IE resolution is dynamically changed early in the simulation. The resolution begins at 100 μs IE resolution to set initial conditions, and then is relaxed to 10 ms at simulation time 1 ms.

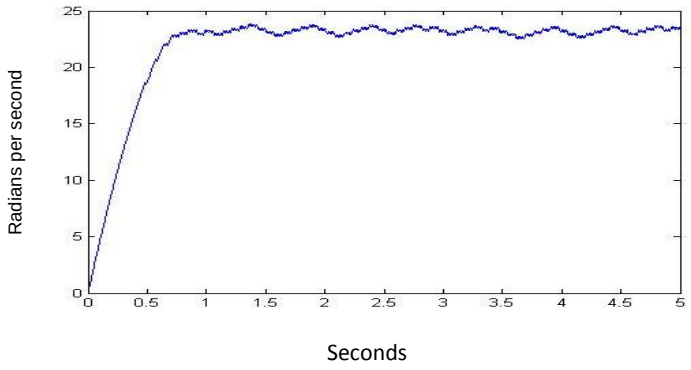


Figure 16. Case C model output speed versus time

Case C shows set point approach, but steady state instability as Simulink and TExaS only receive signal updates every 10 ms (the relaxed IE duration). However, the simulation time decreases significantly as shown in Table 5.

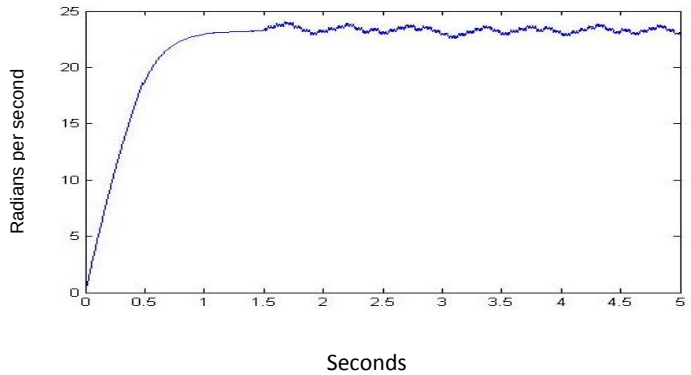


Figure 17. Case D model output speed versus time

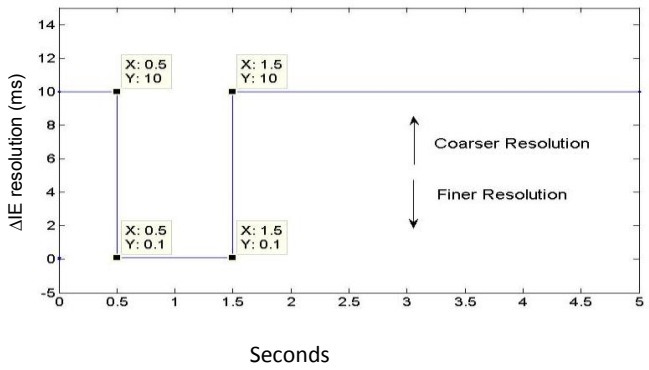


Figure 18. Case D dynamic IE duration change

In Case D, coarse resolution (10 ms IE) is used in the motor run-up phase (0 to 0.5) seconds, then finer resolution (100 μs IE) during the set point approach phase (0.5 to 1.5 seconds), then coarse resolution again in the steady state phase (1.5 to 5 seconds). The control profile matches the static high-resolution case up to 1.5 seconds, where the controller is unstable again in the error due to relaxed resolution (10 ms IE signals). The simulation time is still decreased (Table 5).

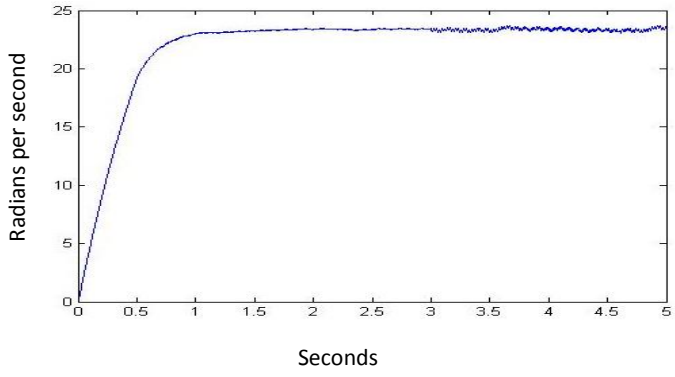


Figure 19. Case E model output speed versus time

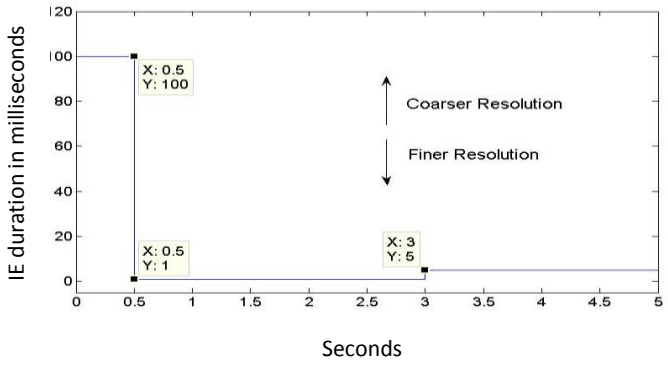


Figure 20. Case E dynamic IE duration change

In Case E, coarse resolution IE (100 ms) is used to 0.5 seconds in the simulation, then finer resolution (1 ms IE) as the set point approaches, and then 5 ms IE resolution in the steady state after 3 seconds. The instability around the set point is reduced over the Case E 10 ms IE resolution after 3 seconds, but variance persists due to controller only getting samples once in every five PID loops (5 ms IE with a 1 kHz PID loop).

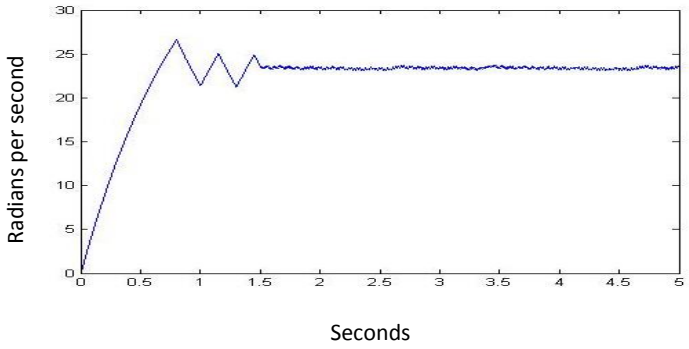


Figure 21. Case F model output speed versus time

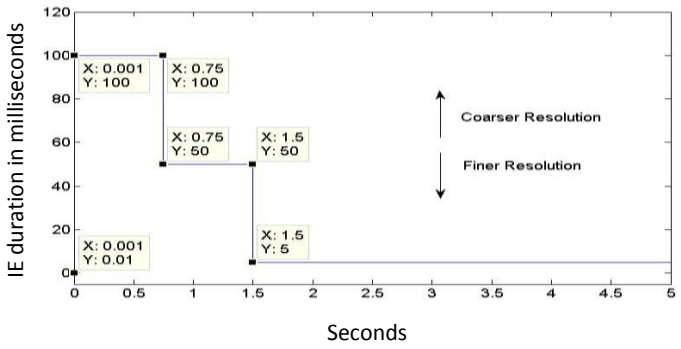


Figure 22. Case F dynamic IE duration change

In Case F, we suppose coarse resolution might apply during the run-up and set point approach phases. Coarse resolution IE (100 ms) from 0 to 0.75 seconds is applied, medium resolution (50 ms IE) from 0.5 to 1.5 seconds, and then 5 ms IE resolution after 1.5 seconds. Figure 21 shows that although the control oscillates coarsely under low resolution, the oscillation decreases significantly when the resolution increases around controller steady state set point.

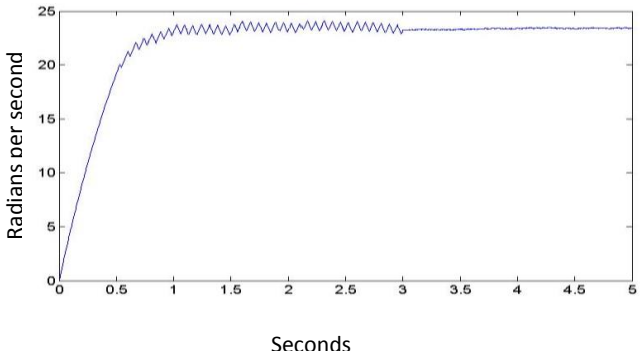


Figure 23. Case G model output speed versus time

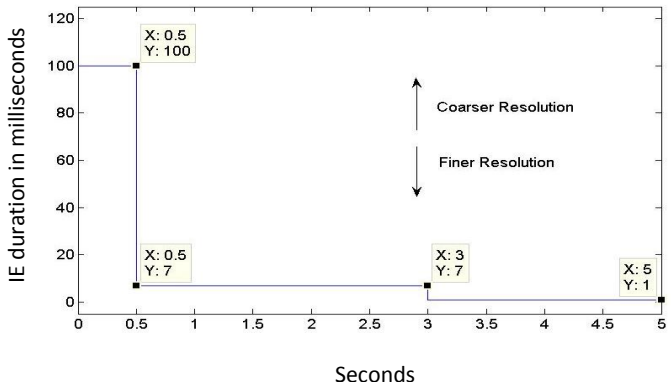


Figure 24. Case G dynamic IE duration change

In Case G, coarse resolution IE (100 ms) from 0 to 0.5 seconds, finer resolution IE (7 ms) approaching the set point, and then 1 ms IE resolution after 3 seconds is applied. The steady state oscillation decreases significantly as the PID controller receives IE updates ever 1 ms. Noise still exists due to 1 ms IE sampling of the PortT[0] PWM wave that has a higher duty cycle resolution than 1 ms. However, Case F shows that we can *arbitrarily* bring down the error term around the set point by increasing the resolution, while using coarse resolution in the run-up and approach phases.

In Figure 25, the speed up multiplier of each case is plotted against the static resolution simulation time of Case A. The maximum percent error of measurement around the set point value of 23.44 after 3 seconds for each experiment is also plotted, with Cases A and B considered the truth condition. Figure 25 shows that percent error around the set point can be brought down arbitrarily while decreasing simulation time by dynamic resolution. In the best trial, Case G, there is a maximum 0.21 percent error of measurement around the set point but with a 6.14 times faster simulation time than the static resolution Case A.

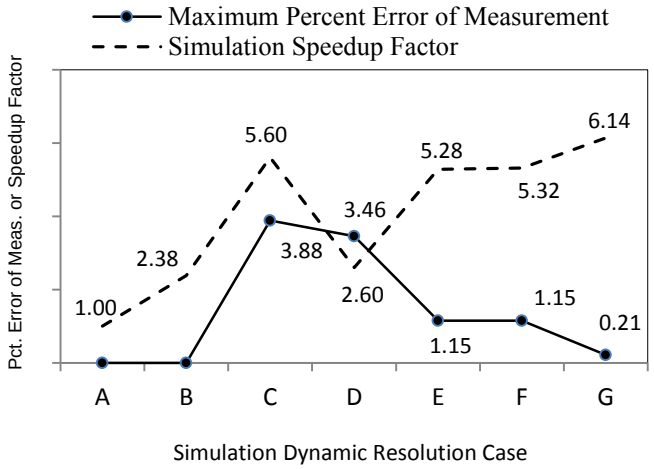


Figure 25. Speedup versus accuracy with dynamic resolution cases

A	B	C	D	E	F	G	H	I
A. S,T,I	100 µs	3	4000	20	11.8	5.0	13:35	2
B. S,T,I	100 µs	3	4000	20	11.8	5.0	5:51	1
C. S,T,I	(dyn)	3	4.8	20	11.8	5.0	2:29	2
D. S,T,I	(dyn)	3	800	20	11.8	5.0	5:21	2
E. S, T,I	(dyn)	3	24	20	11.8	5.0	2:38	2
F. S,T,I	(dyn)	3	6.5	20	11.8	5.0	2:37	2
G. S, T,I	(dyn)	3	19.7	20	11.8	5.0	2:16	2

Table 5. Dynamic resolution case times and counters

Column Legend

A Experiment case and distributed simulators : Simulink(S), TExaS(T), SimConnect server(I)
B Static IE resolution, or (dyn) if dynamic resolution
C Number of instantiated SimTalk connectors
D Number of SimTalk messages (times 10^3)
E Number of 9S12 cycles (times 10^6)
F Number of 9S12 instructions (times 10^6, rounded 10^5)
G Simulated Time (seconds)
H Simulation Execution Time (minutes: seconds)
I Number of host machines

Table 6. Simulation times, configurations and traffic legend

Table 5 shows that distribution (2 machines versus 1 in Cases A and B) affects simulation time as IEs must travel over a LAN. However, even with this cost, dynamic

resolution still decreases simulation time in Cases C through G over the static resolution cases. The SimConnect server in all cases but B ran on GNU/Linux 2.6.16 kernel machine. The TExaS and Matlab/Simulink simulators ran on a Windows 8 OS Intel Core i5 laptop computer.

Dynamic resolution implementation in the SimConnect/SimTalk tools was straightforward with the interpolated event (IE) data type and SimTalk "res" messages. In the space of simulated software-based PID control, these results show that more coarse resolution is tolerable while the PID error value is high, while a higher resolution is required as the controller settles around the set point. This can speed up PID/PWM-based simulations in cyber-physical systems by relaxing resolution when a new controller set point is ordered, then increasing resolution through the steady state of the simulated controller.

6.4 Chapter Summary

The SimConnect and SimTalk KPN-IE method tools enable cyber-physical system simulation of homogenous (many identical simulator) and heterogeneous (many different simulator) systems. For homogeneous simulation, SimConnect and SimTalk enabled the coordination of up to 128 Spice simulators for the parallel speedup of the simulation of a 128 bit digital counter at the transistor circuit level. Interpolated Event resolution versus simulation speed enables a tradeoff in speed versus accuracy of the simulation. For heterogeneous simulation, SimConnect and SimTalk were applied to simulate a software-based PID/PWM controller of a DC-motor. The TExaS simulator modeled a Freescale 9S12-based software PID controller driving a pulse-width modulated (PWM) MOSFET circuit driver in Ngspice, followed by a 2^{nd} order differential equation-modeled DC motor in Matlab Simulink. Tradeoffs in speed with IE

resolution versus accuracy apply, with simulation agreement between the 3-simulator model of system and a 1-simulator model of the system as a truth condition. Finally, means of dynamic runtime-based control of the simulation resolution was explored for the 2-simulator based simulation. Dynamically varying the IE resolution at different time points during the simulation created further increases in simulation speed with tunable tradeoffs in accuracy.

CHAPTER SEVEN. THESIS SUMMARY AND FUTURE WORK

This chapter summarizes important concepts of the KPN-IE method for CPS PADS and the benefits of the SimConnect and SimTalk simulator coordination tools. Additionally, future opportunities for new contributions from the KPN-IE method and SC/ST tools are described.

7.1 Benefits of the KPN-IE Method, Implementation, and Results

Any method that tries to coordinate multiple, independently running, heterogeneous simulators encounters challenges repeatedly seen in the field of parallel and distributed simulation (PADS). These challenges are, but not limited to: simulator messaging (format, content, and carrier), simulator synchronization (keeping simulator events in sync compared to non-parallel execution), simulator event causality (preventing simulators from advancing in time ahead of events that might come from other simulators), simulator messaging deadlock (possible with simulators in zero-lookahead cycles), and simulator deployment (resourcing and distribution).

Two strong PADS solutions that address each of these challenges are the DEVS solution [20] and the IEEE 1516 HLA [19] solution. DEVS offers a unified modeling based approach to PADS challenges. Systems are defined in terms of DEVS set theoretic formalisms, which enable diverse systems to be modeled by virtue of the closure of composition of the DEVS formalisms. DEVS models have continued to evolve from contributions of researchers since the publication of the formalism by Bernard Ziegler in 1976 [20]. DEVS can be described as a unified modeling solution, where interfacing is defined and analyzed at the level of models and models of computation.

Another solution, the IEEE 1516 HLA standard [19], offers an architecture for interfacing diverse simulators, instead of interfacing diverse models over a single

simulator environment. HLA was designed as a U.S. Department of Defense standard for distributed military simulations dating to 1996 [36], and later became an IEEE standard as its utility reached into other domains of simulation [19].

Cyber physical system (CPS) simulation, however, poses challenges to both solutions. First, each component in the CPS may not be already modeled in a DEVS formalism, and there may not be market or research time available to remodel the components in DEVS. Second, simulators in a CPS system may not all have HLA ambassador interfaces written for them to be coordinated by an HLA RTI. The cost of writing and debugging an HLA ambassador for a closed architecture simulator can be a significant market and research cost in a CPS simulation.

A third method, novel to this work, is the Kahn Process Network and Interpolated Event method, or KPN-IE. The heart of the KPN-IE method is to leverage as many inherent dynamics of a dataflow network formalism as possible to provide simulator synchronization and communication services rather than explicitly conducting them through dedicated function calls or additional coordination-dedicated simulator messaging channels. In the KPN-IE method, the signal content and time synchronization content of the simulation is embedded in the content of the KPN dataflow token (the Interpolated Event) rather than in separate explicit signal value and time coordination function calls. The primary advantage of this is that we can hope to reduce simulator interfacing costs, particularly code development costs, by leveraging the inherent synchronizing properties of a KPN with IE tokens. Additionally, we can hope to reduce backplane functionality requirements by making the KPN backplane software primarily an IE token router.

The synchronizing properties of the KPN-IE method were proven in Chapters Three and Four. Namely, the method enforces the local causality constraint (LCC) for

conservatively coordinated simulators, and provides an algorithm for optimistic and combined conservative and optimistic coordination that removes some of the functional burdens from the simulators (such as Time Warp [65] anti-message queues) by offloading them to the KPN backplane. Further, bounds on important coordination values, such as the conservative Lower Bound Time Stamp (LBTS) value and the optimistic Global Virtual Time (GVT) value fall automatically out of managing the KPN IE data streams rather than requiring the backplane to use a dedicated algorithm.

Implementation of the KPN-IE method was achieved with the SimConnect and SimTalk software tools, which coordinated up to 128 Ngspice simulators and 3 diverse simulators (TExaS, Matlab, and Ngspice) for homogeneous and heterogeneous system simulation. These also provided speedup gains through runtime dynamic resolution of IE token values, with tradeoffs in speed versus accuracy. The primary benefits of the SimConnect/SimTalk tools are their code sizes compared to open source HLA implementations. The SimConnect backplane is smaller in code size (up to a third of the code size of the OpenHLA C++ RTI), and the SimTalk plugins for Ngspice and Matlab are significantly smaller in code size than an example open source HLA RTI ambassador for Matlab [79]. It is conjectured that the simplified functional requirements of the KPN-IE method solution versus an HLA solution is the cause of the resulting differences in software code sizes. The code comparisons reflect the hope that interfacing simulators through a SimConnect/SimTalk solution will be less costly in software engineering effort than interfacing them through an HLA RTI.

However, because the HLA RTI is increasingly studied in PADS literature and popularly used and reported in PADS case studies, the KPN-IE approach does not preclude the use of HLA. SimTalk connectors can be interfaced to RTI ambassadors to coordinate a SimConnect/SimTalk simulation with an HLA simulation federation.

In the realm of CPS simulation, which can require the coordination of many diverse engineering design simulators, it is often the case that a simulator is a proprietary solution maintained by professional developers specializing in a field. These simulators may not expose all of their internal software or intellectual property, but may offer model device-level interfaces, the ability to compile OS-system libraries, the ability to set and read simulator signals, and ability to schedule simulator events. These are the only interfaces and abilities a SimTalk IE port connector requires. The connector does not need to tightly interface into the simulator's time advancement kernel. An HLA federate or RTI ambassador, however, requires a federate to surrender all internal simulator time management for time regulated simulation to the RTI backplane. This can prohibit the interfacing of closed architecture CPS simulators for simulation engineers that do not have access to the simulator's proprietary source code. Therefore, the KPN-IE method can enable the interfacing of proprietary simulators without exposing their intellectual property due to the light weight requirements of a SimTalk plugin compared to an HLA ambassador.

Finally, IE dynamics are software friendly. KPN IE streams can be sent to or sourced from databases, piped to diverse consumers, mathematically analyzed, and exchanged between continuous time environments and discrete time environments. Localizing simulator coordination to IE stream dynamics can also be of benefit when the simulation increases in simulated time scale or the number of simulators participating.

7.2 Opportunities

Future contributions are possible with the KPN-IE method for CPS PADS. These opportunities aim to expand the range of CPS PADS simulations or to increase the modeling capability of IE ports through higher order interpolation. Leading suggestions are summarized as follows.

Expanding the number of SimTalk connectors for increased CPS simulation model range

The system component range of SimConnect/SimTalk based CPS simulations can be increased by writing more SimTalk plugins for established and emerging engineering and physical system simulators. This includes, but is not limited to, writing plugins for ModelSim [44], Cadence [25], Synopsis [26], ComSol Multiphysics [45], LabView [46], SystemC-AMS [47], Xyce [48] and other popular, model-rich, domain-specific simulators. As more plugins are written for different simulators, coordination among sets of simulators not previously able to coordinate by their construction alone is enabled. As more plugins are written for different simulators, the combinatorial mix of coordination possible increases on the order of the power set (set of all subsets) of the set of SimTalk-supported simulators, an exponential increase.

Simulating large-scale CPSs with multiple coordinated SimConnect backplanes over a massively parallel cluster such as TACC Ranger [50] or an XSEDE resource [78]

Multiple SimConnect backplanes may be instantiated, each with local clusters of simulators connected to them. This allows assigning one backplane per board on massively parallel machine, with each simulator per backplane running multi-core on the board, similar to a hybrid MPI/OpenMP distributed topology. Instantiating multiple SimConnect/SimTalk networks in this configuration also provides a platform for

researching convergence control and termination techniques in partially-asynchronous, asynchronous, and synchronous parallel iterative numerical methods [41][42]. IEs and KPNs directly map to the "partially asynchronous parallel iterative" approach for solving linear systems and fixed point equations, described thoroughly in [39]. It is possible to examine speed versus convergence tradeoffs in these systems by dynamic control of interpolated event duration in the SimConnect backplane over large data sets.

Connecting SimConnect/SimTalk and HLA

A SimConnect/SimTalk simulation may itself be a federate of an HLA (IEEE 1516) simulation if SimTalk plugins are written for HLA federate and RTI ambassadors. This can bridge a CPS system simulation supported by SimConnect/SimTalk and an HLA-coordinated simulation.

Higher Order Interpolation on Interpolated Events

The Interpolated Event data type may be expanded to include additional signal information, such as the first-order rate of change in the IE value v from t_m to t_n. This provides a first-order interpolation on signals for which the set V is isomorphic to the set of real numbers. First-order interpolation can support adding event quantizers [61] on IE output ports rather than limiting ports to sampling mode or tracking mode.

BIBLIOGRAPHY

[1] Lee, E.A. "Cyber-Physical Systems: Design Challenges." The University of California at Berkeley Center for Hybrid and Embedded Software Systems. Technical Report No.UCB/EECS-2008-8, 2008.

[2] National Science Foundation. "Cyber-Physical Systems." http://www.nsf.gov/pubs/2012/nsf12520/nsf12520.htm. 2006.

[3] Sangiovanni-Vincentelli, A. "Quo Vadis, SLD? Reasoning about the Trends and Challenges of System Level Design." *Proc. IEEE* 95 (2007): 467–506.

[4] Klesh, A.T., Cutler, J.W., and E.M. Atkins. "Cyber-Physical Challenges for Space Systems." In *IEEE/ACM Third International Conference on Cyber-Physical Systems ICCPS*, 2012, 45–52.

[5] Rajkumar, R., Lee, I., Sha, L., and J. Stankovic. "Cyber-Physical Systems: the Next Computing Revolution." In *ACM/IEEE 47th Design Automation Conference DAC*, 2010, 731–736.

[6] Fujimoto, R.M. "Parallel Discrete Event Simulation." In *Proceedings of the 1989 Winter Simulation Conference*, 1989, 19-28.

[7] Chandy, K.M., and J. Misra. "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs." *IEEE Trans Software Eng* (1979): 5.

[8] Jefferson, D.R. "Virtual Time." *ACM Trans Program Lang Sys* (1985): 7.

[9] Amory, A., Moraes, F., Oliveira L., Calazans, N., and F. Hessel. "A Heterogeneous and Distributed Cosimulation Environment." In *Proceedings of the 15th Symposium on Integrated Circuits and Systems Design*, 2002, 115–120.

[10] Pfeifer, D., and J. Valvano. "Kahn Process Networks Applied to Distributed Heterogeneous HW/SW Cosimulation." In *Electronic System Level Synthesis Conference ECSI*, 2011, 1-6.

[11] Pfeifer, D., Valvano, J., and A. Gerstlauer. "SimConnect and SimTalk for Distributed Cyber-Physical System Simulation." *Simulation: Transactions of the Society of Simulation and Modeling International SCS* 89, no. 10 (2013): 1254-1271.

[12] Schmerler, S., Tanurhan, Y., and K.D. Muller-Glaser. "A Backplane Approach for Cosimulation in High-Level System Specification Environments." In *Proceedings of the European Design Automation Conference EURO-DAC* '95 *with EUROVHDL*, 1995, 262–267.

[13] Atef, D., Salem, A., and H. Baraka. "An Architecture of Distributed Cosimulation Backplane." In *42nd Midwest Symposium on Circuits and Systems*, vol. 2, 1999, 855–858.

[14] Sung, W., and S. Ha. “A Hardware Software Cosimulation Backplane with Automatic Interface Generation.” In *Proceedings of the Asia and South Pacific Design Automation Conference ASP-DAC*, 1998, 177–182.

[15] Gheorghe, L., Bouchhima, F, Nicolescu, G., and H. Boucheneb. “Formal Definitions of Simulation Interfaces in a Continuous/Discrete Cosimulation Tool.” In *Proceedings of the Seventeenth IEEE International Workshop on Rapid System Prototyping*, 2006, 186–192.

[16] Bouchhima, F., Briere, M., Nicolescu, G., Abid, M., and E.M. Aboulhamid. “A SystemC/Simulink Cosimulation Framework for Continuous/Discrete-Events Simulation.” In *Proceedings of the 2006 IEEE International Behavioral Modeling and Simulation Workshop*, 2006, 1–6.

[17] Pfeifer, D., and A. Gerstlauer. “Expression-level Parallelism for Distributed Spice Circuit Simulation.” In *15th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications DS-RT*, 2011.

[18] Pfeifer, D., Valvano, J., and A. Gerstlauer. “Dynamic Resolution in Distributed Cyber-Physical System Simulation.” In *Proceedings of the 2013 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation SISIM-PADS*, 2013, 277-284.

[19] IEEE Std 1516-2010. “IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)–Framework and Rules.” 2010, 1–38.

[20] Zeigler, B. *Theory of Modeling and Simulation*. 2nd ed. San Diego: Academic Press, 2000.

[21] Nutaro, J. *Building Software for Simulation: Theory and Algorithms, with Applications in C++*. New Jersey: Wiley, 2011.

[22] Dong, H., Dong, W., and Y. Ji. “A HLA-based Hierarchical Architecture for the CTCS Hardware-in-the-Loop Simulation System.” In *The 2nd IEEE International Conference on Computer Science and Information Technology*, 2009, 86–91.

[23] Fujimoto, R.M., and R.M. Weatherly. “Time Management in the DoD High Level Architecture.” In *Proceedings of the 1996 10th Workshop on Parallel and Distributed Simulation*, 1996, 60–67.

[24] de Mello, B.A., and F.R. Wagner. “A Standardized Co-simulation Backbone.” In *The 11th International Conference on Very Large Scale Integration of Systems-on-Chip*, 2001, 121–131.

[25] Cadence Design Systems, Inc. www.cadence.com. 2012.

[26] Synopsys, Inc. www.synopsys.com. 2012.

[27] Zhou, Y., and M. Wang. “Launch Vehicle Testing Simulation System.” In *The 2011 International Conference on Computational and Information Sciences*, 2011, 921–924.

[28] Crowley, P. “A Dynamic Publish-Subscribe Network for Distributed Simulation.” In *The 22nd Workshop on Principles of Advanced and Distributed Simulation*, 2008, 150.

[29] Kahn, G. “The Semantics of a Simple Language for Parallel Programming.” *Inf Process* (1974): 471-475.

[30] Valvano, J. *Embedded Microcomputer Systems: Real Time Interfacing*. 3rd ed. Stamford, CT: Cengage Learning, 2011.

[31] Nenzi, P., and V. Holger. “Ngspice Users Manual.” vol. 22. ngspice.sourceforge.net. 2010.

[32] The MathWorks Corp. www.mathworks.com. 2012.

[33] Cox, F.L., Kuhn, W.B., Li, H.W., Murray, S.D, Tynor, S.D., and M.J. Willis. “Xspice User’s Manual.” Computer Science and Information Technology Laboratory, Georgia Tech Research Institute, 1992.

[34] Narayanan, R., Abbasi, N., Zaki, M., Al Sammani, G., and S. Tahar. “On the Simulation Performance of Contemporary AMS Hardware Description Languages.” In *The 2008 ICM Tnternational Conference on Microelectronics*, 2008, 361–364.

[35] Lee, E.A., and A. Sangiovanni-Vincentelli. “Comparing Models of Computation.” In *The IEEE/ACM International Conference on Computer-Aided Design Digest of Technical Papers ICCAD,* 1996, 234–241.

[36] Fujimoto, R.M. *Parallel and Distributed Simulation Systems*. New York: Wiley, 2000.

[37] Franklin, G., Powell, J.D., and A. Emami-Naeini. *Feedback Control of Dynamic Systems*. 4th ed. New Jersey: Prentice Hall, 2002.

[38] Hughes, A. *Electric Motors and Drives: Fundamentals, Types and Applications*. 3rd ed. Oxford, UK: Elsevier, 2006.

[39] Bertsekas, D., and J. Tsitsiklis. *Distributed and Parallel Computation: Numerical Methods*. Bemont, MA: Athena Scientific, 1997.

[40] Lee, E.A. “Cyber-Physical Systems – Are Computing Foundations Adequate?” Position Paper for the NSF Workshop on Cyber-Physical Systems: Research, Motivation, Techniques, and Roadmap. 2006.

[41] Bahi, J., Contassot-Vivier M., Couturier R., and F. Vernier. “A Decentralized Convergence Detection Algorithm for Asynchronous Parallel Iterative

Algorithms." *IEEE Transactions on Parallel and Distributed Systems* 16, no. 1 (2008).

[42] Smith, S., and H. Krad. "A Parallel, Iterative Method For Solving Large Linear Systems." In *IEEE Southeastcon Proceedings*, 1990.

[43] Larsen, E. "Admittance Measurement for Assessment of Cardiac Hemodynamics in Clinical and Research Applications." PhD Diss., The University of Texas at Austin, 2012.

[44] Mentor Graphics, Inc. www.mentorgraphics.com. 2013.

[45] Comsol, Inc. www.comsol.com. 2013.

[46] National Instruments, Inc. www.nationalinstruments.com. 2013.

[47] SystemC-AMS. www.systemc-ams.org. 2013.

[48] Xyce Parallel Electronic Simulator. www.xyce.sandia.gov. 2013.

[49] Texas Advanced Computing Center/ACES Visualization Lab. http://www.tacc.utexas.edu/resources/visualization. 2013.

[50] Wang, X., Turner, S., Low, M., and B. Gan. "Optimistic Synchronization in HLA-Based Distributed Simulation." *Simulation: Transactions of the Society of Simulation and Modeling International SCS* 81, no. 4 (2005): 279-291.

[51] Nutaro, J., and H. Sarjoughian. "Design of Distributed Simulation Environments: A Unified System-Theoretic and Logical Processes Approach." *Simulation: Transactions of the Society of Simulation and Modeling International SCS* 80, no. 11 (2004): 577-589.

[52] Pawletta, S., Drewelow, W., and T. Pawletta. "HLA-based Simulation within an Interactive Engineering Environment." In *Proceedings of the Fourth IEEE International Workshop on Distributed Simulation and Real-Time Applications DSRT*, 2000.

[53] Quaglia, F., Santoro, A., and B. Ciciani. "Towards Transparent Optimistic Synchronization in HLA." In *The 2005 Workshop on Techniques, Methodologies, and Tools for Performance Evaluation of Complex Systems*, 2005.

[54] Thepppaya, T., Tandayya, P., and C. Jantaraprim. "Integrating the HLA RTI Services with Scilab." In *The Sixth IEEE International Symposium on Cluster Computing and the Grid CCGRID*, vol. 2, 2006.

[55] Sung, C., and T. Kim. "Framework for Simulation of Hybrid Systems: Interoperation of Discrete Event and Continuous Simulators Using HLA/RTI." In *The 2011 IEEE Workshop on Principles of Advanced and Distributed Simulation PADS*, 2011.

[56] Roth, C., Almeida, G.M., Sander, O., Ost, L., Hebert, N., Sassatelli, G., Benoit, P., Torres, L., and J. Becker. "Modular Framework for Multi-level Multi-device

MPSoC Simulation." In *The 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Ph.D. Forum IPDPSW*, 2011.

[57] Permulla, K., Park, A., and V. Tipparaju. "GVT Algorithms and Discrete Event Dynamics on 129k+ Processor Cores." In *The 2011 18th International Conference on High Performance Computing HiPC*, 2011.

[58] Liu, B., Yao, Y., Jiang, Z., Yan, L., Qu, Q., and S. Peng. "HLA-based Parallel Simulation: A Case Study." In *The 2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation PADS*, 2012.

[59] Nutaro, J. "On Constructing Optimistic Simulation Algorithms for the Discrete Event Specification." *ACM Transactions on Modeling and Computer Simulation TOMACS* 19, no. 1 (2009): 1-21.

[60] Yilmaz, A., Jin, J., and E. Michielssen. "A TDIE-Based Asynchronous Electromagnetic-Circuit Simulator." *IEEE Microwave and Wireless Components Letters* 16, no. 3 (2006).

[61] Nutaro, J. "Discrete Event Simulation of Continuous Systems." Handbook of Dynamic Systems Modeling. 2005.

[62] Zhang, C. "Integrating Existing DEVS Simulations With The HLA." Master's Thesis, Carleton University, 2004.

[63] Nichols, K., Kazmierski, T.J., Zwolinski, M., and A.D. Brown. "Overview of SPICE-like Circuit Simulation Algorithms." *IEEE Proceedings on Circuits, Devices and Systems* 141, no. 4 (1994).

[64] Ferenci, S., Permulla, K., and R. Fujimoto. "An Approach for Federating Parallel Simulators." In *Proceedings of the Fourteenth Workshop on Parallel and Distributed Simulation PADS*, 2000.

[65] Jefferson, D., Beckman, B., Wieland, F., Blume, L., and M. Diloreto. "Time Warp Operating System." *Proceedings of the Eleventh ACM Symposium on Operating Systems Principles SOSP* 21, no. 5 (1987): 77-93.

[66] Nutaro, J. "A Discrete Event Method for Wave Simulation." *ACM Transactions on Modeling and Computer Simulation TOMACS* 16, no. 2 (2006): 174-195.

[67] Lungeanu, D. "Distributed Simulation of Digital and Analog VLSI Systems." PhD Diss., The University of Iowa, 2000.

[68] Lam, S., and A. Shankar. "A Relational Notation for State Transition Systems." *IEEE Transactions on Software Engineering* 17, no 7 (1990).

[70] Keller, R. "Formal Verification of Parallel Programs." *Communications of the ACM* 19, no. 7 (1976): 371-384.

[71] Broy, M., Jonsson, B., Katoen, J-P., Leucker, M., and A. Pretschner, eds. *Model-Based Testing of Reactive Systems: Advanced Lectures*. Heidelberg: Springer, 2005.

[72] Olderog, E., and H. Dierks. *Real-Time Systems: Formal Specification and Automatic Verification*. Cambridge, UK: Cambridge University Press, 2008.

[73] Rewienski, M. "A Perspective on Fast-SPICE Simulation Technology." In *Simulation and Verification of Electronic and Biological Systems*. Dordrecht: Springer, 2011.

[74] Kuhl, F., Weatherly, R., and J. Dahmann. *Creating Computer Simulation Systems: An Introduction to the High Level Architecture.* NJ: Prentice Hall, 1999.

[75] Ghosh, S., and T. Lee. *Modeling and Asynchronous Distributed Simulation: Analyzing Complex System*. New York: IEEE Press, 2000.

[76] Turner, P. *Guide to Scientific Computing*. 2nd ed. Florida: CRC Press, 2001.

[77] Golub, G., and J. Ortega. *Scientific Computing: An Introduction With Parallel Computing*. San Diego: Academic Press, 1993.

[78] Extreme Science and Engineering Discovery Environment (XSEDE). www.xsede.org. 2013.

[79] Stenzel, C., and S. Pawletta. MatlabHLA Toolbox. http://www.mb.hs-wismar.de/~stenzel/software/MatlabHLA.html#GetMatlabHLA. 2013.

[80] CERTI open source HLA RTI. http://savannah.nongnu.org/projects/certi. 2013.

[81] ONERA French Aerospace Lab. www.onera.fr. 2013.

[82] OpenHLA. http://ohla.sourceforge.net. 2013.

[83] ON Semiconductor Corp. www.onsemi.com. 2013.

[84] International Rectifier Corp. www.irf.com. 2013.