Proceedings of the 2013 Winter Simulation Conference
R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, eds.

Interacting Real-Time Simulation Models and Reactive Computational-Physical Systems

Hessam Sarjoughian
Soroosh Gholami

Arizona Center for Integrative Modeling & Simulation
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University
Tempe, AZ 85281, USA

ABSTRACT

For certain class of problems notably in cyber-physical systems it is necessary for simulations to be indistinguishable from computational-physical systems with which they interact. Hard real-time simulation offers controlled timing which lends it to be composed with systems operating in physical time. Accurate real-time simulation equipped with distinct input/output modularity for simulation and software systems is proposed. To demonstrate this approach, a new model for composing ALRT-DEVS (a hard real-time simulation platform) with computational-physical systems is proposed. It provides an abstract communication model for hard real-time simulation and software systems to interact. Experiments are developed where a simulated control switch model (with single and multiple inputs and outputs) operates four independent mechanical relays. In another experiment, the control switch commands are communicated with software capable of triggering simulated and mechanical relays in real-time. In this setting, simulated software, physical, or computational-physical systems may be interchanged with their actual counterparts.

1 INTRODUCTION

Systems of systems have been experiencing rapid rise in their complexity, particularly from the standpoint of timing (Jamshidi 2010). For many years conceptualization, development, operation, and even retirement of these systems are supported with logical-time simulations. However, integral to some of these systems is the necessity to operate in physical time. For example, a modern vehicles operations are controlled via software executing under (soft and/or hard) real-time constraint.

In some cyber-physical systems (Derler et al. 2012), interactions taking place among software and physical systems require both physical time and abstract time. Two abstract representations of physical time are logical-time and real-time. The former is independent of computational platform in which time is handled. The accuracy of the latter is subject to hardware. A simplified representation of the logical-time is event sequence in which passage of time is represented as a total-ordered set of steps. From theoretical perspective, logical-time (or real-time) is represented as discrete or real values with infinite accuracy. From implementation perspective, both logical-time and real-time have finite accuracy.

Common simulations of cyber-physical systems are specified using logical-time and/or event sequence abstractions. A variety of theories and techniques can be used to model and simulate these systems. Both logical-time and event sequence representations of the physical time are useful. Theoretical models can be specified and manipulated with arbitrary accuracy for inputs, outputs, and states. Concrete simulations can only approach the theoretical accuracy defined in the model abstractions. In general, computational precision afforded in host software/hardware system is assumed to have insignificant side effect on the simulation results. Indeed, abstraction of physical time as either logical-time or event sequence in defining abstract models and simulators are ubiquitous in numerous scientific and engineering application domains.

For some systems the abstractions for the physical time pose major limitations. Execution of a simulation in terms of linear time or event sequence may not be a part of systems of systems operating in physical time. Consider a simulated cruise controller for a real vehicle. The simulation during execution must receive sensor signals just-in-time from actual sensors placed on wheels. Sensor data must be detected and transmitted to the cruise controller neither too fast nor too slow given the physical characteristics of the vehicle. The simulators execution speed must be within some acceptable lower and upper time bounds given a wheels angular speed.

Physical operations of a system may be processed using real-time simulation. This is possible assuming a real-time model can be executed in a simulator having a real-time clock. The simulator must be able to correctly execute the model (i.e., consume inputs and produce outputs in real-time). Unlike physical time which has infinite precision, real-time used in the model and in the simulator can have finite accuracy (resolution). The hardware can guarantee timing precision up-to a finite threshold. Real-time simulations although not as precise as logical-time simulation can serve purposes that are not achievable otherwise.

As noted above, the major use of real-time simulation of a sub-system is that it can be synthesized with a physical sub-system. The system as a whole runs in mixed real-time and physical-time. The input and output interactions taking place between the virtual and physical entities generally occur synchronously. This of course assumes exactly the same amount of time passed in the real-world also passed in the virtual world. This requires formulating a concise synchronization across physical and real-time. This requirement can be satisfied if the hardware and software operations needed to execute the simulation can sustain the speed at which the physical and software sub-systems are operating at (e.g., simulated cruise controller execution and the wheel operation remain synchronized in time throughout a designated time period).

A sub-system can also be a computational entity such as the cruise controller mentioned above. Similar to the synthesizing of real-time simulation and physical entities, virtual and software sub-systems as well as software and physical entities have their own timing relations. This view holds under the assumption that time is explicitly accounted for in simulation but not in software (i.e., software executes as fast as possible by its hardware). Thus, simulators, software, and physical entities can be considered as three types of sub-systems. A physical system can be computationally-based or not. A pure physical system can be a vehicle wheel. A computational-physical system can be a sensor with A/D and D/A converters.

In this paper, we focus our attention to the synthesis of real-time simulations and computational-physical systems. We use Action-Level Real-Time DEVS simulator which supports hard real-time execution of DEVS models. We use this simulation platform to show its capabilities for interacting with computational-physical systems. The chosen computational-system is a 4-relay Phidget. It is a hardware/software system with four electrically-controlled relays where each can be independently turned on or off using firmware and software. The result is a basic testbed for formulating interactions taking place between virtual and actual systems. We propose a novel proof-of-concept approach for integrated hard real-time simulations and computational-physical systems. Example scenarios are developed to evaluate timing properties of interacting independent action-level simulation operations with software/hardware entities.

2 BACKGROUND

Cybernetics as the foundation for control of mechanical and electrical systems has played a key role in building many of past and existing systems. In recent years, computing has grown to be central to operations of many time-critical systems. Real-time operation of a vehicle cruise controller is but one of numerous systems that must operate in mixed physical time and real time. To analyze and design these kinds of systems real-time simulation is in use. Real-time design requirements for hardware and software parts of a cyber-physical system can be determined at reduced cost and risk.

An example is the Simscape model where real-time simulation is used as a virtual counterpart of a physical system (Miller and Wendlandt 2010). Real-time means driving and observing the simulation input/output in the physical world. Design details such as device actuation and sensor data collection frequencies can be analyzed. Formulation of a design is enabled by Simulink. Solvers with fixed-cost execution and fixed-step size solvers can be used to obtain an acceptable tradeoff between accuracy and speed. Real-time simulation (a sub-system) may then be synthesized with other physical sub-systems.

2.1 Software/Hardware Real-time Control

In the area of multi-processor systems, real-time scheduling has been well studied (Davis and Burns 2011). These algorithms are expected to guarantee a set of tasks representing a running application meet their individual and collective time constrictions. Tasks can have fixed priority. Scheduling algorithms for homogeneous computing systems are divided into periodic and sporadic task models. In the former, tasks arrive periodically in strict order. Tasks may be synchronous or asynchronous. In the latter tasks may arrive independently at any time once a minimum time interval has elapsed since the arrival of the last task. Each task is characterized by its relative deadline, worst-case execution time, and minimum inter-arrival time. Given these, three levels of constraints on task deadlines are implicit, constrained, and arbitrary. Other key concepts are processor load and processor demand which are used to determine whether or not an application's tasks are feasible. There and many formulations can be used to define performance metrics. Real-time scheduling algorithms can be analyzed algorithmically and empirically given resource availability.

The ideas and methods developed for multi-processor architectures are applied to co-design of real-time control systems (Cervin and Eker 2005). Controllers may be designed for optimal performance given limited CPU resources. In this work and other similar to it algorithms are implemented as real-time software executing on hardware with high clock resolution. A hard real-time simulator as described below offers an alternative platform for development of controllers.

2.2 Action-Level Real-Time DEVS

Classical and parallel DEVS formalisms specify simulation models using abstract time (Zeigler, Kim, and Praehofer 2000). ALRT-DEVS (Sarjoughian and Gholami 2013, Gholami and Sarjoughian 2012) was introduced to support the specification of models with real-time constraints using independent, time-constrained, and action-based external and internal transition functions. The definition of atomic ALRT-DEVS models–inspired from RT-DEVS (Hong, Song, Kim, and Park 1997) and Real-time Statecharts (Giese and Burmester 2003)–is as follows: $\langle X, Y, S, A, \Gamma, \Omega, \psi, \lambda, ta \rangle$. Elements X, Y, and S are input, output, and state sets respectively. The action set (A) with use of time-window is defined in RT-DEVS. Internal transition (Ω), external transition (Γ), and activity mapping (ψ) functions are defined using the concept of locations in real-time Statecharts as well as parallel DEVS formalism. Statechart locations for ALRT-DEVS models are uniquely identifiable by the current phase and the remaining time to the deadline. Actions are mapped to each location with a predefined sequence of execution and guarded statechart transitions between locations are incorporated to provide dynamic decision making capability. Making use of locations and guarded transitions in real-time statechart enables the model to adaptively decide on the actions to execute and the ordering among them. Coupled models in ALRT-DEVS do not specify time as in atomic models. The input/output couplings of coupled models are timeless and therefore messages traversing couplings consume zero time with respect to the simulation protocol. Zero time operations or instantaneous message transfers may become problematic in real-time simulation (see Section 2.2.1).

2.2.1 Bounds on Simulation Accuracy

ALRT-DEVS lends itself for developing temporally constrained models for dynamical systems. These time-constrained models are specified using actions, time windows, locations, and the priorities defined for all atomic models. Timing for every action has a continuous time-base with infinite accuracy. However, having a real-time model does not lead to real-time simulation. The actual implementation of the model introduces limitations on timing subject to the underlying platform which executes the simulation models. The constraints introduced by the executing platform define the maximum accuracy of the simulation. This

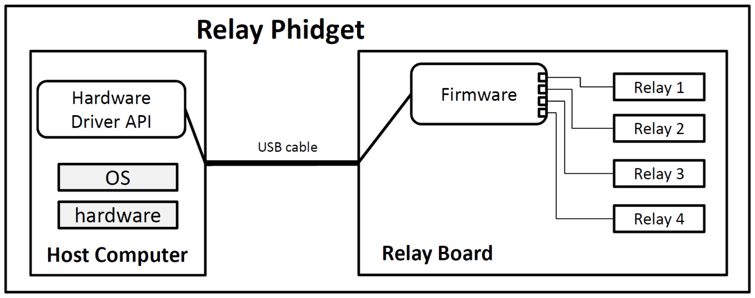


Figure 1: Relay Phidget as a computational system

applies to real-time guarantees as well as time granularity. In other words, the capacity of the executing platform for the load it can manage to execute legitimately (no deadline loss) is bounded. Similar to time resolution, real-time guarantees are not specified in the model and only come into play when the models are realized on a specific platform.

Also, as mentioned in the previous section, in real-time simulation–with regards to the level of time granularity the executing platform supports–instantaneous communications introduce time inconsistency. This inconsistency is usually handled in atomic models by accounting for the real time it takes for messages to reach their destination after departing from the source model. Similar to time granularity and real-time guarantees, time inconsistency–as a result of instantaneous message transfers–only appears during simulation execution.

2.3 Relay Phidget System

For the proof of concept experiments, the PhidgetInterfaceKit 0/0/4 (PhidgetInterfaceKit-0/0/4 2013) (4 digital output ports) is selected. This is a computational-physical system with mechanical relays, firmware, and USB cable being the physical part and the Hardware Driver API being the computational part. This is very simple cyber-physical system where its computational part is un-timed and the physical part executes in physical time. Although the 4-relay Phidget only contains four digital outputs, it is possible to simulate bi-directional communication between the cyber and the physical sides by leveraging the callback signal received from the hardware. These callback signals simply inform the driver (cyber side) of the change in the value of each output port. Therefore, the resulting cyber-physical system is capable of bi-directional interaction where operations in the physical part can be executed under hard real-time constraints.

2.3.1 SW/HW Scope of the Phdiget

A representation of the Phidget as a computational physical system is depicted in Figure 1. As shown in the figure, the Phidget contains both hardware and software sides. In the hardware side, the board contains the communication chip and the relays. The software side consists of an API (Phidget driver) which enables software/simulation to interact with the Phidget relays.

The operation of the physical side of the Phidget is based on electrical current and electromagnetic fields created by the flow of current in the relay coil. The mechanical process of switching in each relay limits the operational frequency of the system. Despite the limitations on operational frequency, the relays are considered as pure physical systems, which perceive time as continuous (infinite time resolution). On the other hand, the software side is untimed (sequenced) or has an abstract concept of time (logical or real) with a certain resolution based on the underlying platform. Therefore, the Phidget in its entirety is considered a discrete time system based on the platform its software side is executed on. The underlying platform–illustrated in Figure 1 as hardware and operating system layers–is shared among all tasks running on the software platform. Usually, the impact of the hardware on the API is neglected and the interaction with the board is considered platform independent or instantaneous.

On the other hand, real-time simulation is heavily influenced by the executing platform. Since all interactions between the cyber and physical components occur in physical time, experimentations (data collection and analysis) should also occur in real-time. This adds to computational load on the underlying platform (software and hardware), which may ultimately result in loss of accuracy (both in experimentation and simulation). This is extensively discussed in (Gholami and Sarjoughian 2013) in which this point is clarified that unlike logical-time, real-time experimentation cannot be treated trivially and requires system/platform dependent design.

3 COMPOSING SIMULATION AND COMPUTATIONAL-PHYSICAL SYSTEMS

The systems of systems discussed above may be categorized to three types. *Physical systems* are operated without the aid of software systems. A mechanical relay used in the relay Phidget is a very simple example of physical systems. As a Single Pole, Double Throw switch, the relay can be controlled with AC and DC power sources. Another physical system is an engine of a vehicle built from mechanical and pneumatic sub-systems. A software system has both software and hardware. In the Phidget the mechanical relay activation (turning on or off) is operated via a software system which consists of an interface API executing on a personal computer. Combining a physical system and a software system results in a *computational-physical system* (e.g., see the Relay Phidget shown in Figure 1). The term computational is restricted to software-based computations even though some biological, mechanical, electrical circuits or others may offer capabilities similar to those of software systems. Physical, software, or computational-physical systems are real-world entities. Computational-physical systems are cyber-physical systems if the former can offer the capabilities designated for the latter.

Simulation systems are a special kind of software systems where their dynamics are governed in artificial settings. In this work, simulation systems are considered to be strictly virtual entities where there are no real world entities. This kind of simulation is akin to constructive simulation where all parts of a system of systems are simulated (DoD 1995)). Simulations involving real people operating real systems (called live simulation (DoD 1995)) are not considered.

A class of systems of systems has simulation and computational-physical systems. These systems are akin to virtual simulations (DoD 1995) where simulated systems are operated by real people. Interactions between simulation and computational-physical systems can be viewed as those within computational-physical systems. However, as noted earlier there are important distinctions between software and simulation systems. Abstractions in simulations, grounded in abstract time, offer flexibilities that are not usually considered in software systems unless time critical operations are required. Inherent to simulation is the ability to arbitrarily accelerate or slow down its execution which is to say the virtual dynamics of a system is artificially structured and governed.

A major distinction between simulation and software systems is worth noting here. In approaches such as DEVS a simulation model is defined to have a specification and an abstract protocol which defines the order in which the parts of a model are to be executed. Software system operations and executions, however, are generally not separated. Most software systems execute based on best-effort principle where there is an underlying assumption that sufficient resources are available for implemented system. Thus integrating hard real-time simulation with best-effort software systems needs special attention. Hard real-time used in simulation can be used for software integration.

3.1 A Model for Composing ALRT-DEVS Simulation with Physical Systems

Hard real-time simulations and computational-physical systems can be synthesized in different ways. Simulation systems can interact with software systems. Simulations can also interact with physical systems if simulation and software systems are indistinguishable. Since simulation and software systems are distinct types of systems, simulation systems can interact with physical systems via software systems. In the case of the Phidget system, a simulation model can operate the mechanical relay via the Phidgets software.

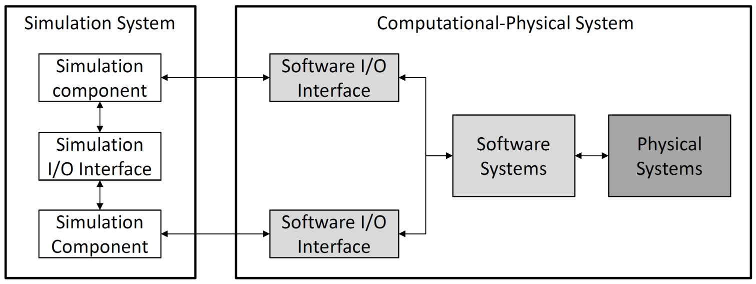


Figure 2: Composing ALRT-DEVS Simulation models with the computational-physical system

A logical, structural model for composition of simulation and computational-physical systems is illustrated in Figure 2. In the simulation system, components interact via simulation I/O interfaces (ports). Messages sent and received among model components are communicated via these I/O ports. The simulation interfaces shown in Figure 2 sanction simulation component to/from simulation component communications according to a modeling formalism and its associated simulator protocol.

Modularity for I/O simulation and software interfaces is proposed. Communications between simulation components and software components are sanctioned via software interfaces. A typical method is for the simulation components to interact directly with software components (Hu and Zeigler 2005). For example, direct call can be made inside external, internal, output, or helper functions of a DEVS simulation model running as a standalone simulation. Direct calls can pass data (output) to a software component. Direct calls can also retrieve data (input) from software for use in simulation. In distributed setting, I/O operations can be synchronous and asynchronous.

Simulation components may communicate with software components in logical-time and real-time. The execution speed of the computational-physical systems has no impact on logical-time simulation. The correctness of hard real-time simulation is crucial for computational-physical systems. Outputs of the simulation component must be produced and delivered to the software components within some specified hard time limits. Similarly, simulation components must receive inputs from the software components within designated hard time windows. Resource availability, therefore, affects real-time simulation dynamics in relation to the computational-physical system with which it interacts.

Inaccuracy in dynamics of hard real-time simulations can be due to two reasons. First the timings defined for operations in simulation model components are inaccurate. This is not considered in this work. Second the simulator is unable to execute the models operations before their deadlines. Separating simulation to software communication helps in reducing execution inaccuracy. Simulation models can be developed and evaluated more accurately since communication with software components is excluded. This can be significant for simulations requiring frequent and high volume communications with software components. A less visible benefit is that simulation system I/O modularity in the simulation is strongly preserved.

As shown in Figure 3, simulation input/output ports are exclusive to simulation model components. Software input/output ports (UML 2.0 (OMG 2004)) are defined for communication with software system. Outputs generated by atomic model components can only be sent to software components via the highest-level coupled model. The same requirement holds for software outputs destined for atomic model components. Each simulation model can have multiple simulation and software input and output ports. The basic concept of strict modularity as defined in parallel DEVS is enforced. Hierarchical external software input coupled to the software input ports of atomic model component is defined. Similarly, atomic simulation model component outputs coupled to the external software output ports is defined.

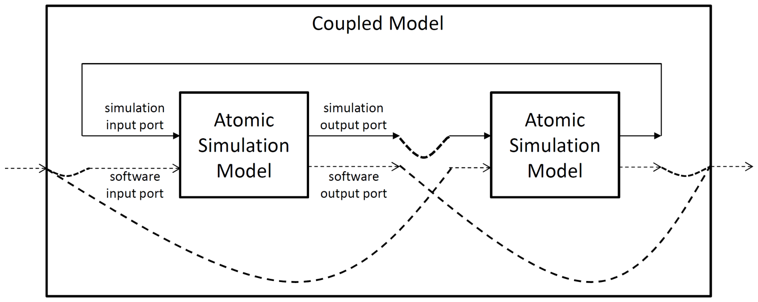


Figure 3: Interaction layers between the real-time simulation and the physical system

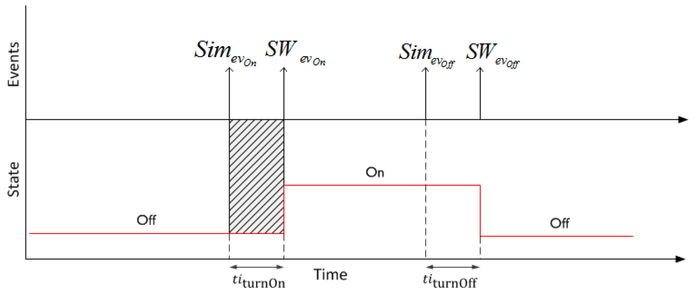


Figure 4: Behavior of the simulated relay and its physical counterpart

3.2 Hybrid DEVS-Suite/Phidget

DEVS-Suite simulator is extended to support the class of hybrid simulation and software systems modeling and execution as described above. Atomic and coupled ALRT-DEVS model specifications enable bi-directional simulation to software communication. The DEVS Simulation Hardware Abstraction (DSHAL) is responsible for input/output data conversion and handling for software systems. It acts as a generic proxy between the DEVS-Suite simulation engine and software systems. Its operations are writing to and reading from the simulation component output and input ports. Interaction with specific computational-physical systems is mediated through designated hardware-specific software. Hardware Driver (i.e., PhidgetInterfaceKit kernel library) is an example of such hardware-specific software. Functions such as *memSet()* belonging to the kernel library executes invocation calls initiated in the simulation model components. For activating the mechanical relays as well as querying their states as stored in the Hardware Driver, DEVS-Phidget software has been developed. The DEVS-Phidget functions are listening for input and output events. Operations related to the Hardware Driver include reading/writing to/from memory and attaching/detaching relays.

A sample event trajectory between simulation and the relay is illustrated in Figure 4. The trajectory is shown from the perspective of the physical/simulated relay. The events are either generated within simulation or the relay. The phase trajectory shows handling of events with *turnOn*/*turnOff* actions. Time duration to execute operations in the DSHAL and DEVS-Phidget can be considered negligible. Real-time simulation granularity is in milliseconds whereas execution of a function in these non-real time software is much smaller by comparison. On the other hand, some operations in the Hardware Driver can consume as much real-time as the simulator. This is due to the mechanical operations taking place in the relay (minimally 30ms) and I/O with the host computer's OS. Depending on the computational-physical system, ALRT-DEVS may not be able to complete all of its actions within a time period.

The separation of the software system from DEVS-Suite simulator allows collecting and time stamping software events and simulation events. In the centralized setting, DEVS-Suite Simulator, DSHAL, DEVS-

Phidget, and Hardware Driver are executed in the Java VM. This is advantageous since JVM clock can be used for all.

4 PROOF-OF-CONCEPT EXPERIMENTS

In this section, several experiments are designed and carried out to serve as proof of concepts for the discussions made above. In these experiments, the data of interest is the time it takes for the relay to switch from on to off or vice versa when triggered by the simulated generator. Since there are no means to measure the time when the relay is switched either on or off, the timestamp is recorded when the callback signal is received in the cyber side from the physical relay. Therefore, the switch-on/off times are in reality the aggregate time of the control signal leaving the cyber side up to receiving the callback signal from the hardware (signal turnaround time). Since turning off and on are identical from the point of view of the simulation and timing, in these set of experiments, we did not distinguish between these two operations. Also, in all experiments with multiple options, one is chosen randomly such as choosing two relays to toggle.

All experiments are done on a Windows 7 computer with Intel Core 2 Due 1.86 GHz processor and 2 GB of memory. The models are executed on DEVS-Suite 2.1.0 and Java 7. Three experiments are designed in this sections that are described below.

4.1 Single vs. Fan-out Signaling

In this experiment, the difference between single hardware port manipulation and fan-out signaling (multiport manipulation) is under inspection. Phidget hardware is known to support parallel logic. This experiment is aimed at recognizing whether our real-time simulation, hardware adapters (DSHAL), and Phidget driver–overall, the cyber side–can leverage this parallelism and manipulate relays concurrently.

4.2 Single and Multi I/O Systems

SISO and MIMO atomic models are designed to support hardware interaction using primitive data types to eliminate the need for data transformation from complex object to physical signals, which in this case is electrical current. SISO/MIMO schemes suppose to simplify the interface between cyber and physical sub-systems. For this experiments two new base classes are developed which support interaction using *double* data type.

4.3 Mixed Physical and Simulated Relay Execution

In this experiment, the Phidget is transformed from pure physical system to a mixed simulated/physical system. For this purpose, 3 hardware relays of the physical Phidget are coupled with a single simulated relay inside the cyber sub-system. In order to imitate the behavior of the physical relays, the timing data gathered from the hardware ports are fed into the simulated relay as its timing configuration.

Figure 5 depicts the outline of the mixed physical/simulated relay along with the adapter (DSHAL). As illustrated in the figure, signals produced by the generator (aimed at simulated relay or physical relay), all travel the same path to the I/O interface (DSHAL). However, from there, simulated relay signals are fed back to the outer coupled model to reach the simulated relay while hardware relay signals are forwarded to the hardware driver. The cardinality of each port is specified in the figure. In order to further mimic the operation of the physical relays, the simulated relay also sends a callback signal to the DSHAL informing the receipt of signal and state change. Therefore, both generator and simulated relay are simulation models which communicate indirectly through DSHAL. Also, timing data (both simulated and hardware relays) is still gathered in DSHAL.

Figure 6 is a part of the simulated relay realization in DEVS-Suite. In the external transition function in line 8, on/off signals are received from the input port and change the phase of the relay. The simulated

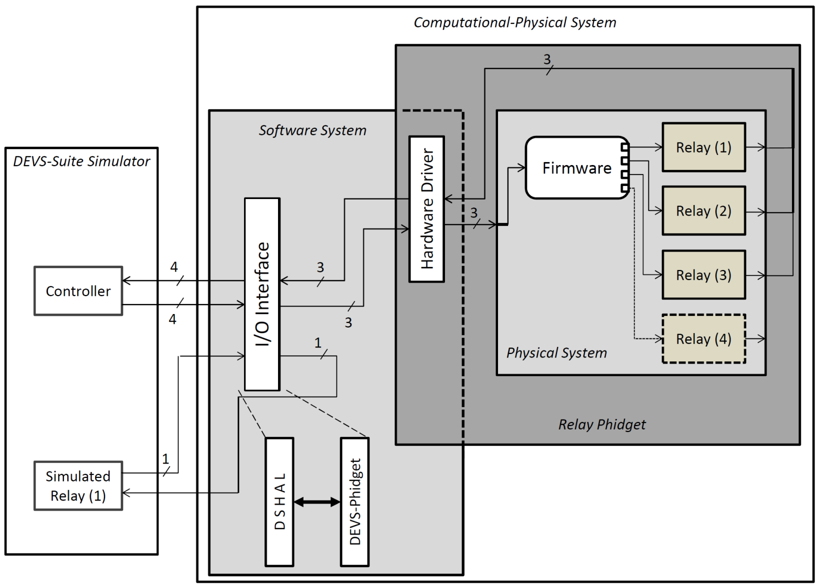


Figure 5: Mixed simulated/hardware Relay Phidget scheme

relay has two actions: one for switching off and one for switching on (not included in the code snippet). As mentioned above, the time duration for executing these two actions is read from a configuration file and stored in the data structure *switchingSpeed*. This data is then used for specifying the duration of switching on and switching off actions in lines 13 and 17, respectively. The output function in line 1 is responsible for sending callback signals to the DSHAL as described above. Communication between DSHAL and simulated/physical relays is done by *hwBooleanEntity* which stores a boolean value, the origin of the signal, and the port with which the signal is transmitted.

4.4 Results

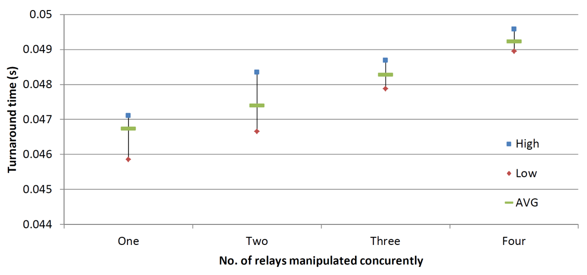
For the first experiment (Single vs. Fan-out Signaling) a multiport generator is modeled and simulated based on ALRT-DEVS with a single output port. This generator is capable of sending complex objects (port numbers and boolean values) to the adapter. The adapter then discretizes these complex objects into basic hardware signals and transmits them to the hardware via the Phidget driver. Experiments are done for 1 up to 4-port manipulation, to see the level of parallelism supported by the cyber side. Each experiment consists of 50 signals and repeated 5 times. As shown in the results in Figure 7-a, there is a millisecond time difference between one, two, three, and four relay manipulation. Therefore, this proves that although the hardware side supports parallel port manipulation, the software side is incapable of such task since the callback signals are received one by one and serviced sequentially. However, this shortcoming is at the level of software instead of the real-time simulation. Although the simulation side sends signals as bundle, it receives the callback signals sequentially which shows sequential processing in the software layer (Phidget driver). The diagram shows a steady rise in the numbers and decrease in the difference between the highest and the lowest turnaround time.

As for the second experiment (Single and Multi I/O Systems) other than the difference in data transformation which was mentioned above, SISO and MIMO operate quite similar to regular single and fan-out signaling models. The results retrieved from the SISO experiments were similar to those of single port manipulation in the previous experiment. In MIMO, at every cycle, a random number between 0 and 4 (both inclusive) is generated which specifies the number of relays to be manipulated and then the state of the selected relays are altered. As expected, the behavior of all hardware ports are similar. The stock chart in Figure 7-b demonstrates the results retrieved from this experiment. The differences observed in

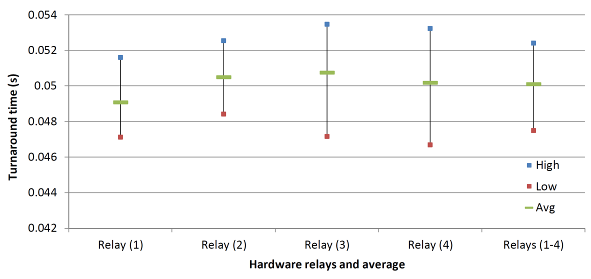
```
public message out() {
   message m = new message();
   m.add(makeContent("outport",
         new hwBooleanEntity(this.state, this, "outport")));
   return m;
}

public void deltext(double e, message x){
    Continue(e);
    for(int i = 0 ; i < x.getLength() ; i++){
      hwBooleanEntity m = (hwBooleanEntity)x.getValOnPort("inport", i);
      if(m.getv()){
        turnOn.setExactExecutionTime(switchingSpeed(true));
        phase = "turnOn";
      }
      else{
        turnOff.setExactExecutionTime(switchingSpeed(false));
        phase = "turnOff";
}}}
```

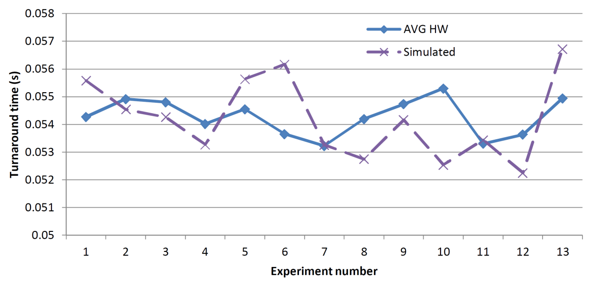
Figure 6: External transition and output functions for the simulated relay in mixed simulation scheme



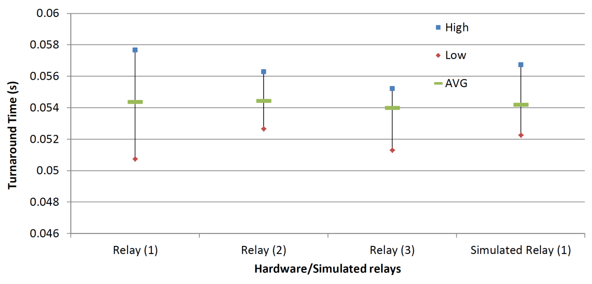
a) Phidget relays operated with single vs. fan-out messaging



b) Phidget relays with MIMO messaging



c) Hybrid simulated and actual Phidget relays with MIMO messsageing



d) Hybrid simulated and actual Phidget relayes with MIMO messaging

Figure 7: Experimentation results

the behavior of various hardware ports are the result of unpredictable behavior of the software and the mechanical part of the relays.

In the third experiment (Mixed Physical/Simulated Relay Execution), A total number of 13 experiments are carried out on the mixed relay system in which each experiment consists of 50 send/receive signals. Figure 7-c is a comparison between the average timing behavior of physical relay and the behavior of simulated relay. The timing differences are small and similar to the timing differences between physical ports. Figure 7-d visualizes this data in the form of stock chart with maximum, minimum, and median data for each relay. The simulated port shows similar behavior (similar median and lower/upper bounds) to the other hardware ports.

5 Discussion

Simulation has demonstrated its use in many areas including support for embedded (software/hardware) system design. An example is the use of logical-time DEVS simulation for autonomous, intelligent cruise controller (Schulz et al. 1998). Software/hardware co-design is approached as a process where appropriate tradeoffs can be studied in a virtual setting to implement the cruise controller. Soft real-time DEVS simulation is used for analyzing and designing cooperative simulated and actual robots (Hu and Zeigler 2005). In these and other DEVS-based approaches (e.g., Cellier and Kofman (2006), Bergero and Kofman (2011)), simulation model operations are not subject to hard real-time execution and without support for action-level timing. Interactions internal and external to simulation models are not separated as proposed in this work.

Researchers in the domain of cyber-physical systems have also developed model-based design and simulation approaches and tools. These works describe challenges (e.g., timing among interacting parts of CPS, scalability, and reuse) that face development of cyber-physical systems. For example, design contracts where integration of systems can be evaluated in (logical-time and soft real-time) simulated settings is proposed and shown using Ptolemy II (Derler et al. 2013). In another effort, SystemC with timing annotation is used for virtual prototyping of the network needed for software systems to control physical systems (Zhang et al. 2013). This work proposes simulation as an attractive alternative to hardware-in-the-loop simulation.

Another trend is to utilize platforms supporting real-time software execution. Use of dedicated hardware and platform-specific programming languages afford virtualizing realistic dynamics of physical systems. Simulation models developed in high-level programming languages may be transformed to programming languages that can execute using real-time operating systems. For example, Ptolemy II models developed in Java programming language can be converted to hardware-specific C language. This requires representing the model syntax and semantics to match target programming language (Giotto) and OS (FreeRTOS). Other works transform DEVS models to VHDL (e.g., Pifer (2012), Molter et al. (2009)). Embedded system simulation is has also been proposed (e.g., Yu and Wainer (2007)). The importance of hard real-time simulation of physical systems has also attracted researchers to use special-purpose platform implemented. An example is the use of FPGA instead of general-purpose computing platforms for simulating rigid body systems (Bishop et al. 2000). There exist research in distributed, real-time simulation which fall outside the scope of this paper.

6 CONCLUSION

This paper describes a kind of systems of systems where hard real-time simulations and computational-physical systems are mirror images of one another. Toward this goal, simulations interactions within are separated from those with computational-physical systems. A simple scenario is developed where a control switch executes in the ALRT-DEVS platform while interacting with a Relay Phidget. This role of simulation is complementary to the logical-time systems of systems simulations and different types of hardware-in-the-loop simulations used in cyber-physical systems research. With new advances in computation and hardware platforms, accuracy and scale of real-time simulation will increase and thus seamless interaction between simulated and real-world systems should become more achievable. To this end, research in distributed real-time simulation with high fidelity (with action level timing) and simulation model transformations for target computation and hardware platforms are among many exciting future research directions.

REFERENCES

Bergero, F., and E. Kofman. 2011. "PowerDEVS: a tool for hybrid system modeling and real-time simulation". *Simulation* 87 (1-2): 113–132.

Bishop, B., T. P. Kelliher, and M. J. Irwin. 2000. "Hardware/software co-design for real-time physical modeling". In *ICME*, Volume 3, 1363–1366. IEEE.

Cellier, F. E., and E. Kofman. 2006. *Continuous system simulation*. Springer.

Cervin, A., and J. Eker. 2005. "Control-scheduling codesign of real-time systems: The control server approach". *Journal of Embedded Computing* 1 (2): 209–224.

Davis, R. I., and A. Burns. 2011, October. "A survey of hard real-time scheduling for multiprocessor systems". *ACM Comput. Surv.* 43 (4): 35:1–35:44.

Derler, P., E. A. Lee, M. Törngren, and S. Tripakis. 2013. "Cyber-Physical System Design Contracts".

Derler, P., E. A. Lee, and A. S. Vincentelli. 2012. "Modeling Cyber–Physical Systems". *Proceedings of the IEEE* 100 (1): 13–28.

DoD 1995. *Modeling and Simulation (M & S) Master Plan*. Department of Defense.

Gholami, S., and H. Sarjoughian. 2012. "Real-time Network-on-Chip Simulation Modeling". In *SIMUTools, Desenzano, Italy*, 103–112. ICST.

Gholami, S., and H. Sarjoughian. 2013. "Observations on Real-time Simulation Design and Experimentation.". In *SpringSim Multi-Conference*, 1–8. SCS.

Giese, H., and S. Burmester. 2003. "Real-time Statechart Semantics". *TechReport tr-ri-03-239, University of Paderborn*.

Hong, J., H. Song, T. Kim, and K. Park. 1997. "A real-time discrete event system specification formalism for seamless real-time software development". *Discrete Event Dynamic Systems* 7 (4): 355–375.

Hu, X., and B. P. Zeigler. 2005. "A simulation-based virtual environment to study cooperative robotic systems". *Integrated Computer-Aided Engineering* 12 (4): 353–367.

Jamshidi, M. 2010. *Systems of systems engineering: principles and applications*. CRC Press.

Miller, S., and J. Wendlandt. 2010. "Real-Time Simulation of Physical Systems Using Simscape". *MATLAB News and Notes*.

Molter, H. G., A. Seffrin, and S. A. Huss. 2009. "DEVS2VHDL: Automatic transformation of XML-specified DEVS Model of Computation into synthesizable VHDL code". In *Forum on Specification & Design Languages*, 1–6. IEEE.

OMG 2004. "2.0 Superstructure Specification". *OMG, Needham*.

PhidgetInterfaceKit-0/0/4 2013. "Products for USB Sensing and Control". http://www.phidgets.com/.

Pifer, T. J. 2012. "DEVS-Based Hardware Design, Synthesis, and Power Optimization Using Explicit Time Specifications and Deterministic Path-Based Latency". Master's thesis, University of Arizona.

Sarjoughian, H., and S. Gholami. 2013. "Action-level Real-time DEVS Modeling and Simulation. Submitted". *ACM Transactions on Modeling and Computer Simulation*.

Schulz, S., J. W. Rozenblit, M. Mrva, and K. Buchenriede. 1998. "Model-based codesign". *Computer* 31 (8): 60–67.

Yu, Y. H., and G. Wainer. 2007. "eCD++: an engine for executing DEVS models in embedded platforms". In *Proceedings of the summer computer simulation conference*, 323–330. SCS.

Zeigler, B. P., T. G. Kim, and H. Praehofer. 2000. *Theory of Modeling and Simulation*. 2nd ed. Orlando, FL, USA: Academic Press, Inc.

Zhang, Z., E. Eyisi, X. Koutsoukos, J. Porter, G. Karsai, and J. Sztipanovits. 2013. "Co-Simulation Framework for Design of Time-Triggered Cyber Physical Systems".