High-Level Specification and Modeling of Cyber-Physical Systems

José M. Garrido
College of Computing and Software Engineering
Kennesaw State University
1100 South Marietta Parkway
Marietta, Georgia 30060, USA
jgarrido@kennesaw.edu

ABSTRACT
Software development for cyber-physical systems is different and more complicated than conventional computer-based systems, this is because of the hybrid nature of these systems and their complex general properties. This paper proposes a simplified approach to higher-level formal specification and modeling of cyber-physical systems as a way to raise the level of abstraction of the development of these systems. The specification uses a timed temporal logic, high-level modeling approach based on Communicating Real-Time Machines (CRSM), and object-oriented simulation. The modeling includes process interaction with discrete-event simulation using OOSimL language.

This paper includes brief discussion of the specification, modeling, and simulation of timing constraints, concurrency, synchronization, in addition to other expected behavior. The Train-Gate System is presented as a case study that shows the specification of the timing constraints, modeling, and the simulation aspects of this study. The results of the simulation display the reaction times, response times, throughput, and the deadlines missed.

Keywords
Cyber-physical systems, real-time, formal Specification, simulation models, concurrency.

1. INTRODUCTION
A cyber-physical system (CPS) is an integration of computational subsystem with a physical process (physical plant) for the purpose of controlling the physical process. These systems require a combination of computing, communication, and control technologies. Embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa. The study of a CPS must include the interactions of the physical and the cyber parts of the system. Thus, design of CPS involves the modeling of the joint dynamics of these parts. Developing a CPS is very different than developing more conventional computer-based system.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ACMSE'2019 ACM Southeastern Conference, Kennesaw, Georgia, USA
Copyright 2019 ACM X-XXXX-XX-X/XX/XX ...$15.00.
The general specification of CPS must include:

- abstraction and decomposition
- Programming principles
- Data structures
- Algorithm structures
- Specification
- Programming principles and programming languages
- Concurrency and synchronization
- Modeling and simulation
- Principles of embedded and real-time systems
- Hardware, DSP, and communication.

2. INTER-DISCIPLINARY APPROACH

The challenge in developing a CPS is the application of abstraction and decomposition in modeling techniques across the heterogeneous nature of a typical CPS. Most of the conventional modeling techniques in the various parts are currently incompatible. This is seen when considering that physical processes are conventionally studied with continuous models represented by sets of differential equations, control systems are usually studied with difference equations, and software systems are analyzed with discrete models. With all these types of modeling, different methods of solution are required. Most discussions of development methods of CPS describe the limitations at the implementation levels [1, 6, 7].

We argue that we must apply more emphasis on higher-level abstraction. A very high-level model of the CPS would describe the overall intended structure and behavior of the CPS, without considering how it will be implemented.

Because the problem domain crosses the boundaries of multiple disciplines, an inter-disciplinary development process is required that involves the combining of two or more approaches each from an academic field into one single process. An inter-disciplinary approach crosses traditional boundaries between academic disciplines. With this approach, separate discipline approaches are integrated into a single analysis and a common understanding and holistic view of all aspects of the CPS would be achieved.

The term multi-dimensional abstraction is proposed in order to emphasize the need to improve the development of models with an interdisciplinary approach. Every member of the team must use abstraction to reduce the complexity of the real system to fit into the conceptual model. A team member will use his/her own skills and knowledge to determine the essential aspects of the system to model, within that part of the system that relates to their field. Figure 2 illustrates the general multi-disciplinary and inter-disciplinary requirements of modeling.

Figure 2: Inter-disciplinary approach to modeling a CPS.

The general specification of CPS must include:

- Hierarchy
- Compositional behavior
- Timing behavior
- State-oriented behavior
- Event-handling
- Concurrency
- Synchronization and communication
- Presence of programming elements as concurrent tasks
- Executability
- Support for the design of large systems
- No obstacles for efficient implementation
- Domain-specific support
- Non-functional properties

3. REAL-TIME BEHAVIOR OF A CPS

A cyber-physical system has often the goal of controlling a physical process (or part of it). In simpler cases, a cyber-physical system is a supervisory system if it only takes measures (input signals and data) from the physical process at specified time intervals and/or when specified periodic events occur. In this case, the response of the cyber-physical system is limited to record such inputs from the physical process.

Systems with Real-time behavior respond (or react) in a timely fashion to variety of events from the physical process (or physical plant). A computer system usually maintains an on-going interaction with the physical process. In this interaction, real-time tasks are bounded by the specified time windows for their actions. In other words, the response from the computer system cannot occur too early, or too late, given a set of inputs from the physical process.

An embedded system is part of a larger system, it is very a specialized subsystem to suit a specific purpose and does not usually interact with human users or operators. Cyber-physical systems normally include a combination of hardware and software. The range of systems span from small embedded systems to very large and complex systems that include use of wide area networks.

Real-time computing is very different from general-purpose computing. Not only are there strict timing constraints but frequently a very limited set of low-capacity (hardware) resources such as: memory, type of processor, disk space, etc. Customized software must interface with a small number of hardware devices, and smart sensors.

In general, cyber-physical systems must be extremely reliable, robust, dependable, and fault tolerant. They are much more difficult to develop than more conventional systems that use general-purpose computing and the cost of development is much higher. The general principles of real-time systems are discussed in several texts, among others are [2, 11].

3.1 Describing Dynamic Behavior

The general characteristics of real-time behavior of cyber-physical systems are:

- Timeliness - the system must perform operations in timely manner;
Temporal operators are:

- **Tnext** - this operator represents a formula that is true if property \( F \) was true for the next \( \tau \) interval of time in the future.

- **Tprev** - this operator represents a formula that is true if property \( F \) has been true for the previous \( \tau \) interval of time in the past.

- **TholdN** - this operator represents a formula that is true if property \( F \) is always true in the past, present, and future.

- **TholdP** - this operator represents a formula that is true if property \( F \) is true at some time instance in the past, after which property \( F_1 \) became true.

- **Tfirst** - this operator represents a formula that is true if property \( F \) will be true for the first time at \( t \) time in the future.

- **Tlast** - this operator represents a formula that is true if property \( F \) was true for the last time at \( t \) time in the past.

- **Teventual** - this operator represents a formula that is true if property \( F \) will be true eventually at some time instant in the future.

Real-time behaviors of cyber-physical systems include non-terminating tasks because of the reactivity characteristics. The input events from the physical plant have no specific order of occurrence. The communication among real-time tasks is normally defined by message passing, which can be synchronous or asynchronous.

### 3.2 Specifying with Temporal Logic

Formal specification of a system is very important, as it precisely describes the required behavior of the system. Several temporal logics have been used for specifying timing aspects of real-time behavior. The following is a simplification of the ones used in the TRIO+ [10] notations. Time-dependent variables may represent directly physical quantities that change with time. A formula is a predicate, which may hold (true) at any point in time.

Temporal operators can be used to express the occurrence of events in the future and/or in the past. The two fundamental temporal operators are: **Tnext** and **Tprev**.

- **Tnext** \((F, t)\) represents a formula that is satisfied at the current time if property \( F \) is true at time \( t \) in the future. \( Tnext(F, t) \) represents a formula that is satisfied at the current time if property \( F \) was true at time \( t \) in the past. The derived temporal operators are:
  - The following operator represents a formula that is true if property \( F \) will be true in all time instants in the future.
    \[ Tnext(A)(F) = \forall t \left( t > 0 \right) \land Tnext(F, t) \]
  - The following operator represents a formula that is true if property \( F \) was true in all time instants in the past.
    \[ Tprev(A)(F) = \forall t \left( t > 0 \right) \land Tprev(F, t) \]
  - The following operator represents a formula that is true if property \( F \) will be true eventually at some time instant in the future.
    \[ TnextE(F) = \exists t \left( t > 0 \right) \land Tnext(F, t) \]
  - The following operator represents a formula that is true if property \( F \) was true at some time instant in the past.
    \[ TprevE(F) = \exists t \left( t > 0 \right) \land Tprev(F, t) \]
  - The following operator represents a formula that is true if property \( F \) was, is, or will be true at some time instant in the past, present, or future.
    \[ Teventual(F) = TprevE(F) \lor TnextE(F) \]
  - The following operator represents a formula that is true if property \( F \) will be true for the next \( \tau \) interval of time in the future.
    \[ TholdN(F, \tau) = \forall t \left( 0 < t < \tau \right) \land Tnext(F, t) \]
  - This operator represents a formula that is true if property \( F \) has been true for the previous \( \tau \) interval of time in the past.
    \[ TholdP(F, \tau) = \forall t \left( 0 < t < \tau \right) \land Tprev(F, t) \]

The real-time requirements of a system involve specifying the deadlines for each type of input. Because of concurrent real-time tasks and variable delays, satisfying the specification is very complicated. For hard real-time systems, missing a single deadline is taken as incorrect (and unacceptable). For soft real-time systems, a few selected deadlines may be missed occasionally.

The general model for cyber-physical systems adopted here is based on Communicating Real-Time Machines (CRSM) presented in [11]. In CRSM all communications are synchronous and via unidirectional channels that connect two processes, thus it is applying CSP semantics [5]. There are several additional important specification languages suitable for real-time systems, such as: TCOZ [9, 4], TRIO+ [10], and RT-uml [2].

In CRSM, **time constraints** are specified in the transitions between states. A timing constraint is a pair of time expressions with the following meaning:

1. The values of the expressions give the lower and upper bounds on the duration of an internal activity (a computation).
The model of cyber-physical systems discussed here has been developed with the maximum parallelism assumption; i.e., every real-time task is modeled as a process that executes in its own processor and no scheduling issues are discussed. A similar graphical notation for modeling real-time systems is RT-UML and appears in several books, such as [2].

3.4 Timeouts in Communications

When a process takes longer than the deadline set for carrying out a communication activity, a timer process is used to interrupt the process. A timer can also be used to reactivate a task that is attempting to communicate with another task. Once interrupted or reactivated, a task can typically proceed to a different activity or be terminated with the activation of various alarms.

In the usual case, one of the tasks, either the sender or the receiver, will have to wait until the other task is available. When the communication is not possible for a specified period, the communication timeout expires and an exception is enabled. Two cases are considered:

- The sender task cannot wait more than \( \text{send\_per} \) time units to communicate with the receiver task.
- The receiver task cannot wait more than \( \text{rec\_per} \) time units to communicate with the sender task.

A timeout can also occur while a task is carrying out an internal activity like closing a gate; this is internal activity or computation. These activities have a specified pair of bounds for the duration of the activity. If the duration of the activity takes longer than the upper bound, a timeout results. If the duration of the activity takes less than the lower bound, a timeout also results.

The communication support among concurrent tasks in a CPS can be of two types:

1. Asynchronous (shared memory, non-blocking, potential race condition)
2. Synchronous (blocking)

4. TRAIN-GATE SYSTEM

This system consists of a one-directional railway track that crosses a road. There is a gate at the crossing, which is lowered and raised under computer control. A short distance from the crossing, there is a sensor that can detect a train approaching the crossing area. Another sensor is located a short distance after the crossing point, which detects trains departing the area.

4.1 General Description

Trains arrive with a mean inter-arrival period given by \( a \), trains take an average of \( b \) time units to pass the gate area and leave. The gate takes a maximum of \( z \) time units to close or open. There are some delays imposed for communication. The system is specified to ensure that the gate closes before the train reaches the crossing. A small number of trains can be in the area at once. The diagram in Figure 3 shows the model of the Train-Gate real-time system.

The general form of a transition is: \( G \rightarrow act[\tau e] \). \( G \) is the guard of the transition, \( act \) is the associated activity, and \( \tau e \) is the timing constraint.

The safety requirement of the system is that the gate is closed whenever there are trains in the area. The liveness requirement is to keep the gate open if there are no trains in the area.

The response time is the time interval that elapses from the instance a train arrives to the instance the gate completes closing. The reaction time is the interval that elapses from the instance the train arrives (input event) until the gate starts to close.

4.2 Modeling with CRSM

The complete systems consists of real-time tasks that are represented as processes and the communication channels defined among these processes. Figure 3 shows the main components (processes, channels, sensors, and actuators) of the model. The following components are defined:

- The process for the train sensor \( \text{EntryS} \) that detects entering or arriving trains
- The process for the train sensor \( \text{ExitS} \) that detects trains departing
- The process for gate control, \( \text{GateP} \).
- The process \( \text{Monitor} \) that keeps track of the number of trains in the area.
- The controller process \( \text{Controller} \)
- The physical sensors and actuators are part of the physical plant

When a train approaches the area, the corresponding sensor causes process \( \text{EntryS} \) to send a \( \text{train-in} \) signal via channel \( \text{trin} \) to the monitor process, \( \text{Monitor} \). Process \( \text{EntryS} \) will also send a signal via channel \( \text{trout} \) to process \( \text{ExitS} \) so that it can expect a train exit in a future time instant. After receiving a signal from \( \text{EntryS} \), process \( \text{ExitS} \) is prepared to receive a signal from the physical sensor when a train leaves. When this occurs, it sends the signal \( \text{rout} \) to process \( \text{Monitor} \). The train arrivals depends on the train inter-arrival time given by \( a \).

Process \( \text{Monitor} \) maintains updated data of the state of the area in order to send an appropriate \( \text{open} \) signal via channel \( \text{cont\_og} \) and a \( \text{close} \) signal via channel \( \text{cont\_cg} \) to process \( \text{Controller} \). This controller process sends its signals to process \( \text{Gate} \) for opening or closing. The controller process indirectly controls the gate with signal \( \text{opengate} \) via channel \( \text{og} \) and signal \( \text{closegate} \) via channel \( \text{cg} \) to the gate process.
The controller process is the most complex one in the model; its functionality is described as follows: After receiving a signal from the monitor process to close the gate, the controller process attempts to communicate with the GateP process. There is a deadline imposed when attempting communication with the gate process. There is another deadline imposed on the gate process, when the interval for the gate to close exceeds $z$ time units.

In a similar manner, the controller process waits for the monitor process with a signal to open the gate, then it sends a signal to the gate for start to open. The main difference is that while the controller process is waiting for the opening of the gate, it may receive a signal from the monitor for immediate closing of the gate. In this case, the controller process must interrupt the activity of the gate process, then signal the gate for closing.

The gate process ignores the $og$ commands when the gate is open and $cg$ commands when the gate is closed. A $cg$ command causes the gate to physically close; taking $z$ time units of time, and an $og$ command opens the gate. Opening also takes $z$ time units of time. When the gate has completely closed (or opened), it sends the $gok$ signal to the controller process. Figures 6 through 8 show the CRSM diagrams of the processes described.

### 4.3 Specifying Real-Time Behavior with Temporal Logic

The specification of timing aspects of the Train-Gate model follows using the temporal expressions described previously. The state of the gate is defined as:

$$\text{state} == \{\text{up}, \text{lowering}, \text{down}, \text{raising}\}$$

There are two commands to control the gate: $raise$ and $lower$. The following formula expresses the property that $\tau$ is the time interval that elapses to move the gate from the $up$ state to the $down$ state.

$$(\text{state} = \text{up} \land \text{lower}) \rightarrow (\text{TholdN}(\text{state} = \text{lowering}, \tau) \land T\text{next}(\text{state} = \text{down}, \tau))$$

Similarly, to express the property that the time that elapses to move the gate from the $down$ state to the $up$ state is $\tau$.

$$(\text{state} = \text{down} \land \text{raise}) \rightarrow (\text{TholdN}(\text{state} = \text{raising}, \tau) \land T\text{next}(\text{state} = \text{up}, \tau))$$

In a first scenario, if the controller sends a $lower$ command to the gate when it is still in the $raising$ state, the gate will take a longer time to finally reach the $down$ state because it has to get to the $up$ state before starting to move downward again. The total time here is denoted by $\sigma$, which is greater than or equal to $\tau$, or expressed as $\sigma \geq \tau$. The property is expressed as:

$$T\text{next}(\text{state} = \text{down}), \tau) \rightarrow (\exists \sigma (T\text{first}(\text{state} = \text{up}, \sigma) \land T\text{next}(\text{TholdN}(\text{state} = \text{lowering}, \tau) \land T\text{next}((\text{state} = \text{down}, \tau), \sigma)$$

In a similar manner, if the controller sends a $raise$ command to the gate when it is still in the $lowering$ state, the gate will take a longer time to finally reach the $up$ state because it has to get to the $down$ state before starting to move upward again. The total time here is denoted by $\sigma$, which is greater than or equal to $\tau$, or expressed as $\sigma \geq \tau$. The property is expressed as:

$$T\text{next}(\text{state} = \text{lowering} \land \text{raise}) \rightarrow (\exists \sigma (T\text{first}(\text{state} = \text{down}, \sigma) \land T\text{next}(\text{TholdN}(\text{state} = \text{raising}, \tau) \land T\text{next}((\text{state} = \text{up}, \tau), \sigma)$$
In a second scenario, if the controller sends a lower command to the gate when it is still in the raising state, the gate will take a shorter time to finally reach the down state because it would interrupt its upward movement and start to move downward. The total time here is denoted by \( \rho \), which is less than or equal to \( \tau \), or expressed as \( \rho \leq \tau \). The property is expressed as:

\[
\begin{align*}
(state = \text{raising} \land \text{lower}) \rightarrow \exists \rho \left( \\
T_{\text{next}}(\text{TholdN}(\text{state} = \text{lowering}, \rho)) \land \\
T_{\text{next}}((\text{state} = \text{down}, \rho), \rho)
\right)
\end{align*}
\]

5. SIMULATION OF REAL-TIME SYSTEMS

The dynamic behavior of a CPS is studied with the process interaction approach of object-oriented simulation. The traces of the simulation runs of a model of a cyber-physical system is useful for the verification of the specification of the system and for better understanding the overall behaviors of the system.

The simulation principles and modelling are presented in [3]. This includes an introduction to OOSimL and discussions of concepts applied to modeling and simulation of several case studies. The following Web page provides several simulation models, the OOSimL simulation library, and documentation:

http://kauweb.kennesaw.edu/~jgarrido/oosimlc

5.1 Communication Timers with OOSimL

A process attempting communication will wait for some specified interval, and then it will check if the sender (or receiver) process is still waiting to communicate. This is implemented in OOSimL using a communication flag \textit{comm\_wait}, a boolean attribute of class \textit{Process}.

When the timer process checks that the sender (or receiver) process is still waiting after the intervals expired, it will interrupt the sender (or receiver) process. This is implemented in OOSimL by dequesuing the sender process from the master queue, or dequesuing the receiver process from the slave queue in the channel \textit{Waitq} object that represents the communication channel.

When the sender (or receiver) process is reactivated, it checks its interrupt level for the value that corresponds to the communication channel. If the interrupt level corresponds, the process carries out a specified activity like, triggering an alarm, turning off a switch, etc.

To simulate the handling of the earliest and latest times that a communication in a specific process, a normal distribution is used to generate a random time to communicate. The random time generated is used as the time of the communication.

The communication is attempted by the process at some instant within the specified interval. From this point in time, the process waits for the other process to communicate. If the communication is accomplished in the specified time window, then there is no problem with deadlines for this communication. If the communication was not possible in the specified interval, then a timeout is flagged, and the process enters a different state. The names of channels are the names of the objects of class \textit{Waitq}.

The communication of an event is just a signal; no duration period is involved. A random time is generated that defines the instance at which the communication is initially attempted by a process.

A receiver process may receive more than one type of message or event, through one or more channels (respectively). In this case, the receiver process is suspended waiting for an incoming event or message; the sender process needs to reactivate the receiver process. After restarting, the receiver process will detect which channel has the incoming communication.

When timing is involved in the communication, the master process checks the earliest and latest times of the sender’s message/event, and then it determines if the communication is possible.

The simulation model of the Train-Gate system consists of eight classes, one for each of the processes described: \textit{Train}, \textit{EntryS}, \textit{ExitS}, \textit{Monitor}, \textit{Controller}, and \textit{GateP}, class \textit{Train} for the overall model, class \textit{Channel}, and class \textit{Message}. The source files for the simulation model of the Train-Gate system are available from the Web page mentioned previously.

5.2 Simulation Run of the Train-Gate System

The complete source code, the output listings, and the full version of this paper can be found in the \textit{cpstrain} folder in the following URL of the web page:

kauweb.kennesaw.edu/~jgarrido/oosimlc

The following is a partial listing of the output of a simulation run:

```
Simulation of Simple Gate-Train system Mon Oct 8 2018
Total trains that arrived: 6
Average system reaction time: 5.72527
Average system response time: 12.4897
Worst reaction time: 34.3516
Worst response time: 74.9381
Number of deadlines missed in closing Gate: 1
Number of deadlines missed when Controller comm Gate: 0

End of simulation Simple Gate-Train system Mon Oct 8 2018
```

The following listing shows a portion of the trace of a simulation run of the Train-Gate system.

```
Simulation of Simple Gate-Train system Mon Oct 8 2018
----------------------TRACE----------------------
 0.000 Entry Sensor waits for train arrival
 0.000 Controller suspends
 238.512 Entry sensor detected train
 249.031 Monitor received signal from entry Sensor
 249.031 (Monitor) Number of trains now: 1
 249.031 Entry sensor sent signal to Monitor
 249.136 Controller attempt receive close signal
 257.481 Controller received close signal
 257.481 Monitor sent close signal to Controller
 257.481 Controller attempt comm Gate
 257.780 Exit Sensor received signal Entry Sensor
```

6. CONCLUSION

Because of the hybrid nature of a CPS, developing such a system is different than developing a more conventional computer-based system and the process is more complex and thus difficult. We propose to raise the level of abstraction in developing a CPS. This has the advantage of providing a clearer and better understanding of the system. We can achieve this by considering only small number of critical features of the CPS, study only relevant behavior, and emphasizing the general overall structure of the CPS.

The basic dynamic behavior of real-time aspects of cyber-physical systems can be precisely described by using high-level specification and modeling. This paper applied temporal logic is applied to specify the timing constraints. Modeling was applied using CRSM and object-oriented simulation modeling. The results of the simulation show the complete sequence of events and the various performance metrics.

Incorporating simulation enhances not only the understanding of the dynamic behavior of cyber-physical systems, but also reinforces other important basic concepts such as concurrency and synchronization. Graphics with animations can further enhance the mentioned advantages of simulation. Finally, the traces of simulation runs help in the verification of the specification of real-time behavior of cyber-physical systems.

7. REFERENCES


