An End-to-End Tool for Developing CPSs from Design to Implementation

Hyejin Joo, Kyoung-Soo We, Seunggon Kim and Chang-Gun Lee[*]
Department of Computer Science and Engineering
Seoul National University
Seoul, Korea
{hjjoo, kswe, sgkim, cglee}@rubis.snu.ac.kr

For a Cyber-Physical System (CPS), the real-time execution must be guaranteed at the design time for the safe and reliable interaction between a Cyber and a Physical System. Thus, simulation method is widely used to verify and validate the behavior of a CPS, in the development process. Commercial tools of today, however, only mimic the functional behavior of the system, not the temporal behavior. Moreover, when the simulation target system is changed, developers have to reconfigure all settings to simulate properly. To overcome this limitation, we introduce our End-to-End Development Tool that can support the functional and temporal co-validation and smooth migration for the change of the simulation target system.

1 Introduction

A Cyber-Physical System (CPS) is a system where a Cyber and a Physical system interact with each other. A cyber system monitors various states of a physical system using sensors. Computing nodes included in the cyber system compute the monitored states and the system uses the computed results to control a physical system using actuators. Thus, the CPS consists of various hardware components such as sensors, actuators and networked embedded computing nodes and hundreds or thousands of software components, specially called tasks which are predefined programs for special purpose, that are executed on the hardware components. Modern vehicle which is an example of CPSs carries more than 70 ECUs (Electronic Control Units) connected by five different networks[2]. These CPSs must execute tasks in the real-time by a unit of tens of micro-seconds to control the physical system deliberately. If a real-time execution fails, the system will encounter a catastrophic situation; a CPS is a mission critical system. Therefore, when building such a complex and real-time executed system, the simulation method is widely used to verify and validate the whole system in the early phase of the development. However, most simulation approaches only mimic the functional behavior[4],[10] and then fail to simulate the real-time behavior of the simulation target system. In order to see the importance of the simulating not only the functional behavior but also the temporal behavior, Figure 1 shows the control performance of steering control algorithm that keeps the vehicle at the center of the lane. In the Figure 1(c), solid line shows the performance achieved by a commercial tool[10] that mimics only functional behavior assuming the purely periodic tasks represented by the Figure 1(a) and it seems to stay stable. However, the dotted line represents the performance achieved by real execution. You can find that the dotted line oscillates unstably since the behavior of the real physical system represented by Figure 1(b) is different from the simulation result, Figure 1(a). Thus, simulation method must consider not only the functional behavior but also the temporal behavior.

[*]corresponding author

Submitted to:
V2CPS 2016

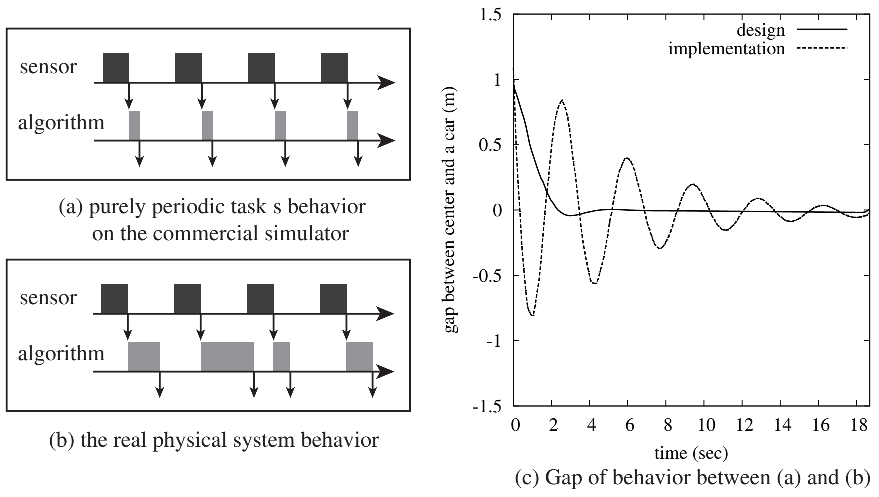

Figure 1: A gap between the simulation result and the real behavior

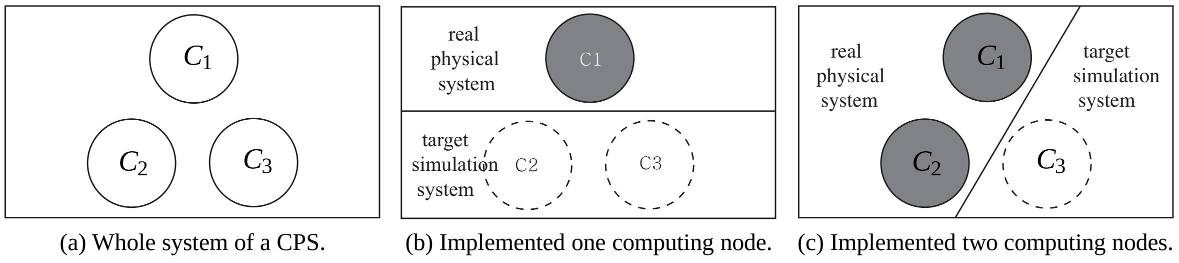

Figure 2: An example of a CPS

Moreover, the simulation target system keeps changing as the development phase progresses. Suppose a CPS that has three computing nodes. Figure 2(a) shows an example of a CPS; C_i represents a computing node to be implemented. Let us look at Figure 2(b). As progressing development process, just one of three computing node, C_1, has been implemented and other two computing nodes, C_2 and C_3, are not implemented yet; these two computing nodes should be simulated, interacting with the already implemented computing node, C_1, in real-time. As time goes by, like Figure 2(c), two computing nodes of the whole system are implemented and just one computing node should only be simulated. That is, the simulation target system has been changed. In this way, when building such large scaled systems, the simulation target system changes frequently. If developers need to reconfigure all settings for the changed simulation target system manually, this process significantly increases the cost and the period of development. Therefore, the development tool must support smooth migration from previous to changed simulation target system when those situations arise.

Motivated by the problems mentioned above, this paper introduces our End-to-End Development Toolchain that supports two concepts: (1) functional and temporal co-validation and (2) smooth migration for the target system changing. We call our tool "D2A-CPSim" which means Design to Auto-implementation for CPS with SIMulator.

2 Related Work

Simulation is widely used for the purpose of validating software system in the early development phase with no real HW platform. Traditionally, a software system has been simulated using cycle-accurate instruction set simulators. Although such simulators provide the cycle-level accuracy, they are too slow for simulating a large system with lots of embedded processors and hence cannot be used for real-time simulation. In order to improve the simulation speed, virtual platform simulators using binary translation are also being used [1],[14]. However, their focus is only fast functional simulation. Thus, they do not care about temporal simulation correctness. Recently, host-compiled simulation drew much attention due to its fast and time-accurate simulation [11],[13]. However, it only targets a standalone device, such as a smart phone without interactions with a physical plant. Thus, it does not consider the correct real-world interactions. In [12], the authors propose an RTOS simulator which executes jobs simulating RTOS scheduling events as we do in our proposed simulator. However, its simulation correctness is not clearly discussed, and furthermore, it is limited to a single embedded processor with an RTOS. Moreover, the authors of [12] do not provide any specific algorithms for executing simulated jobs.

3 Proposed End-to-End Development Tool : D2A-CPSim

As we mentioned in previous sections, End-to-End Development Tool for a CPS must support two key features: (1) functional and temporal co-validation and (2) smooth migration for the target system. In this section, we desire to explain how achieve these two key features on a singlecore computer[1].

To explain smoothly, let us look at Figure 3 showing an example of a Cyber-Physical System; as you know, you can find that a Cyber system and a Physical system interact with each other. This CPS consists of three computing nodes: C_1, C_2 and C_3. Since in the middle of the development process, just one computing node, C_3, might be ready but the other two computing nodes, C_1 and C_2, might be not. We suppose that the computing nodes C_1 and C_2 are in the Simulated World because they have not been implemented yet and should be simulated. Similarly, we suppose that the computing node C_3 is in the Real World because it has already been implemented. As Figure 3 shows, two computing nodes in the simulated world should interact with the real world acting like the real physical computing nodes. In this situation, to support the first key feature, end-to-end tool must simulate not only the functional behavior of computing nodes but also temporal behavior due to events that happen in the target system such as preemptions among the tasks, bus arbitration, etc. with real-world interactions. In addition, to support the second key feature, the tool must have a function that can automatically reconfigure to deal with information about the simulation target system; which computing nodes have been implemented or not, how mapped tasks to computing node, etc..

3.1 Functional and Temporal Co-Validation

In this subsection, we investigate how we achieve the first key feature: Supporting the functional and the temporal co-validation. Let us look at Figure 3 again. As you can see, each computing node is designed to run several tasks represented by τ_i; for example, C_1 will run τ_1 and τ_2 after it is implemented. To simplify, we suppose that the tasks are periodic tasks; tasks are released at periodically predefined time and have constant execution time. These tasks will be run on real physical computing nodes by a scheduling policy that is represented by Figure 4(a). We call the scheduling policy "reference schedule". Since each task

[1]The source code of our tool, that will be explained, is open to the public at [8].

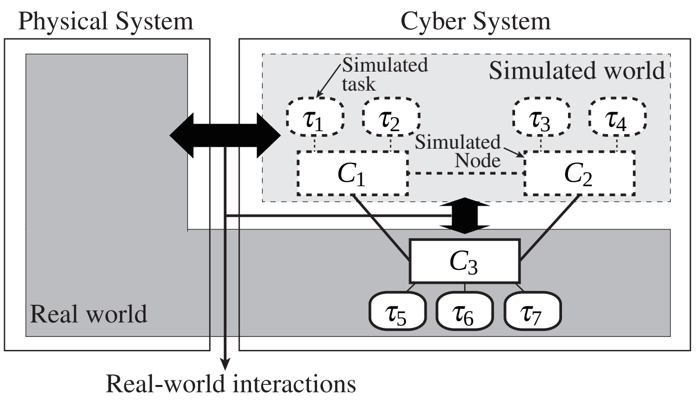

Figure 3: An example of a Cyber-Physical System

should be executed periodically, a unit of tasks' execution is represented by a job; $J_{i,j}$ is jth execution of a τ_i. Now, the problem is how to execute all the jobs in the reference schedule on a single timeline because we assume that the simulation target system should be a singlecore computer. We call this singlecore computer, which can simulation the target system, a simulation host. Note that the execution time of a job on a simulation host, i.e., powerful PC CPU, is much shorter than the expected execution time of the job on its real physical computing node. For the explanation, in this case, let us simply assume that the simulation host is twice as faster as the real physical computing node such that each job's execution time on the simulation host is half of the real physical computing node. Figure 4(b) shows a simple way to dispatch jobs on a simulation host, that is, starting jobs strictly at their expected start times in the reference schedule and emitting the outputs of the jobs strictly at their expected finish times in the reference schedule. Unfortunately, this simple dispatch method does not guarantee correctness of simulation because the several computing node must be simulated on a singlecore computer. Moreover, the simple way cannot use the simulation host efficiently. As we said before, because the simulation host is faster than a real physical computing node, when the simulation of a task is done, the simulation host might be in an idle state. Although a resource, that is simulation host, is usable, it is wasteful to never use the resource. For further explanation regarding this topic, refer to [7].

In short, to co-validate functional and temporal behavior, the simulation target system should be simulated correctly and efficiently.

- **Correct simulation.** From the perspective of the real world, the computing node in the simulated world should yield the right value at the right time to the real world as if the simulated tasks are scheduled exactly the same as the reference schedule.
- **Efficient simulation.** While maintaining the correct simulation, we should make the best use of the simulation host by efficiently executing simulated jobs on the simulation host.

3.2 Smooth Migration for the Simulation Target System Changing

Let us look at Figure 5(a), the same as Figure 4(a). In order to verify and validate the whole system correctly, the functional and temporal behavior of the four tasks, from τ_1 to τ_4, on the simulation host should be the same as if they are executing on the real physical computing nodes, specially interacting with real-world interactions. As time goes by, assume that the C_2 is also implemented as Figure 5(b) showing. Then, real-world interaction points might be changed; the simulation target system must yield

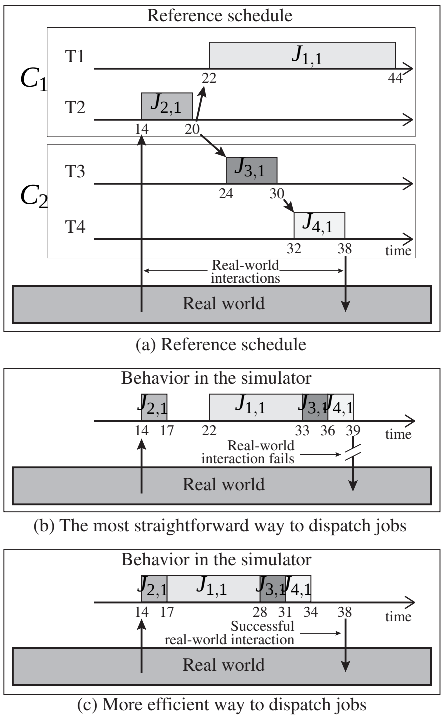

Figure 4: Real-time simulation with real-world interactions

the output at time 24, not at time 38 as before. Therefore, to support smooth migration from the simulation target system to the real physical world, the real-world interaction points must be handled carefully. For the changing real-world interaction points caused by the simulation target system changing, we adopt the concept of the port. The port is a virtualization of the data sending or receiving points. It can be classified into four types: (1) input port from the real world, (2) input port for simulated world, (3) output port to the real world and (4) output port to the simulated world. These ports are the same as below.

- **Input port from the real world.** If a task receives data from the real world, such as τ_2 of Figure 4(a), the task has a input port from the real world.
- **Input port from the simulated world.** If a task receives data from the simulated world, such as τ_1, τ_3 and τ_4, the task has a input port from the simulated world.
- **Output port to the real world.** If a task send data to the real world, such as τ_4, the task has a output port to the real world.
- **Input port from the simulated world.** If a task send data to the simulated world, such as τ_2 and τ_3, the task has a output port to the simulated world.

These ports are used by the tool to reduce the labor of a developer. This process is dealt with more specifically in [15],[6].

4 Comparison with Other Commercial Tools

In this section, we compare to other commercial tools. Our end-to-end development tool supports functional and temporal validation simultaneously, and Table 1 shows differences when using our tool to develop smart car included in CPSs. Each row of the Table 1 is the same as below.

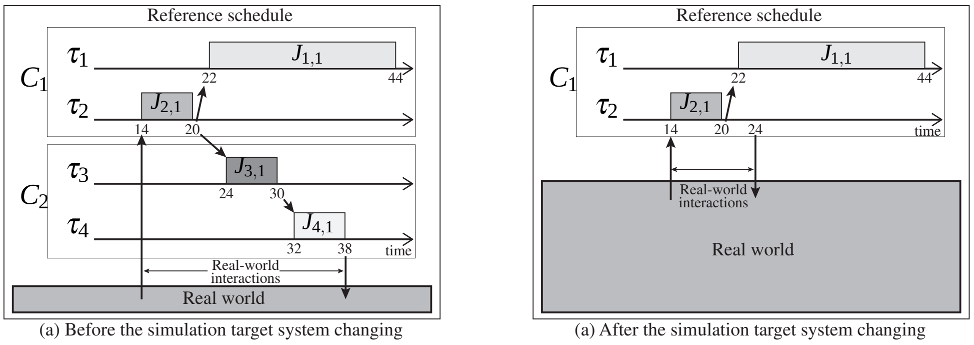

Figure 5: Changed real-world interaction points by the simulation target system changing

Table 1: Comparison with other commercial tools.

	Simulink	Rapid Prototyping	CANoe	ChronSim/ ChronVal	Our End-to-End Development Tool
Functional validation for algorithms	O	O	O	O	O
Validation for internal behavior of ECU	X	X	X	O	O
Validation for CAN behavior	X	X	O	O	O
Validation with real vehicle	X	O	O	X	O
Phased HiLS supporting	X	X	X	X	O

- Functional validation for algorithms: Validate whether the algorithm functions properly.
- Validation for internal behavior of ECU: Validate the real-time behavior of the software components within the ECU.
- Validation for CAN behavior: Validate the real-time transfer of messages through the CAN bus.
- Validation with real vehicle: integrated validation with real vehicle supporting HiLS environment.
- Phased HiLS Supporting: Automatic reconfiguration for each change in simulation environment.

As you can see the Table 1, other commercial tools only support little part of functional validation. Our End-to-End Development tool, however, supports the two key features. From the first row to the fourth row of the Table 1 represent functional and temporal co-validation. In addition, the fifth row of the Table 1 shows any other commercial tool cannot support smooth migration for the simulation target system changing. An example of executing our tool is uploaded at [**?**], so the interested readers are referred to our demo video.

5 Conclusion

This paper introduces our proposed End-to-End Development Tool supporting the key features for CPSs: (1) Functional and Temporal Co-Validation and (2) Smooth migration from the simulation target system changing. To propose our tool, we investigate how these two key features can be achieved in more detail. Functional and temporal co-validation is achieved by correct and efficient simulation on a singlecore computer. Moreover, we adopt the concept of a port to support smooth migration from the simulation target system changing. Lastly, by showing the difference from other commercial tools, only our tool supports the two key features. Thus our tool guarantees that the development process for a CPS is more safe and stable.

In our future work, we plan to extend the introduced tool to support other task types such as aperiodic or event-driven tasks. We also plan to enhance the simulation capacity of our approach by selectively omitting executions of jobs that result in redundant data.

References

[1] Fabrice Bellard (2005): *QEMU, a Fast and Portable Dynamic Translator*. In: *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC ’05, USENIX Association, Berkeley, CA, USA, pp. 41–41. Available at `http://dl.acm.org/citation.cfm?id=1247360.1247401`.

[2] Manfred Broy (2006): *Challenges in Automotive Software Engineering*. In: *Proceedings of the 28th International Conference on Software Engineering*, ICSE ’06, ACM, New York, NY, USA, pp. 33–42, doi:`10.1145/1134285.1134292`. Available at `http://doi.acm.org/10.1145/1134285.1134292`.

[3] C. Andreade C. Dufour & J. Belaneger (2010): *Real-Time Simulation Technologies in Education: a Link to Modern Engineering Methods and Practices*. In: *Proc. of International Conference on Engineering and Technology Education (INTERTECH)*.

[4] National Instrument Corporation: *LabView*. Available at `http://www.ni.com/labview/`.

[5] dSPACE GmbH: *AutoBox*. Available at `http://www.dspace.com/en/ltd/home/products/hw/accessories/autobox.cfm`.

[6] Y. Oh S.-M. Jeong KK.-S. We, J.-C. Kim & C.-G. Lee. (2013): *Demo Abstract: An Efficient and Easilly Reconfigurable Cyber-Physical Simulator*. In: *Proc. of International Conference on Cyber-Physical Systems (ICCPS)*.

[7] Yuyeon Oh Kyoung-Soo We, Jong-Chan Kim & Chang-Gun Lee: *Efficient Simulated Job Execution for Functionally andTemporally Correct Interactions between Simulated World and Real World*. Available at `http://titanium.snu.ac.kr:5002/fbsharing/rpF44ADj`.

[8] Rubis Lab.: *D2A-CPSim (Design to Auto-implementation for CPSs with SIMulator)*. Available at `https://github.com/rubis-lab/D2A-CPSim`.

[9] Rubis Lab.: *A Real-Time Simulator for Automotive Control Systems*. Available at `http://youtu.be/aOOpjzfYOBk`.

[10] Inc. MathWorks: *Simulink*. Available at `http://www.mathworks.com/products/simulink/`.

[11] D. Mueller-Gritschneder, K. Lu & U. Schlichtmann (2011): *Control-Flow-Driven Source Level Timing Annotation for Embedded Software Models on Transaction Level*. In: *Digital System Design (DSD), 2011 14th Euromicro Conference on*, pp. 600–607, doi:`10.1109/DSD.2011.82`.

[12] P. Razaghi & A. Gerstlauer (2011): *Host-compiled multicore RTOS simulator for embedded real-time software development*. In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pp. 1–6, doi:`10.1109/DATE.2011.5763046`.

[13] Sascha Roloff, Frank Hannig & Jürgen Teich (2012): *Fast Architecture Evaluation of Heterogeneous MPSoCs by Host-compiled Simulation*. In: *Proceedings of the 15th International Workshop on Software and Compilers for Embedded Systems*, SCOPES ’12, ACM, New York, NY, USA, pp. 52–61, doi:`10.1145/2236576.2236582`. Available at `http://doi.acm.org/10.1145/2236576.2236582`.

[14] Imperas Software: *Open Virtual Platforms*. Available at `http://www.ovpworld.org`.

[15] K. S. We, J. C. Kim & C. G. Lee (2011): *A novel simulation framework for supporting real-time cyber-physical interactions*. In: *Service-Oriented Computing and Applications (SOCA), 2011 IEEE International Conference on*, pp. 1–3, doi:`10.1109/SOCA.2011.6166237`.