Simulating a Network:

An Approach for Connecting Multiple SystemC Simulations

Thomas W. Pieber, Fikret Basic, Thomas Ulz, and Christian Steger
Institute for Technical Informatics
Graz University of Technology
Graz, Austria
Email: {thomas.pieber, basic, thomas.ulz, steger}@tugraz.at

***Abstract*—Nowadays, sensor networks are widely used. To create new hardware for these networks, simulations are used. These simulations help during the design of the sensor nodes by providing information about internal states, power usage, and expected lifetime. They are useful to design one piece of hardware, however they are cumbersome when multiple of such hardware simulation instances need to interact with each other. This paper explores the possibility of starting multiple instances of a hardware simulation and connecting these via a simulation of the environment they will be used in.**

Keywords–SystemC; Simulation; Parallel; Network.

I. INTRODUCTION

The concurrent development of hardware and software enables the accurate simulation of the behavior of a finished sensor node. Rather than focusing solely on the hardware simulation, languages such as SystemC can also calculate the software influences on the system. This is a necessary step towards the complete simulation of a network. To simulate the behavior of a network, multiple such sensor instances need to be simulated simultaneously. These virtual sensor nodes are then connected via a suitable environment simulation.

There are many solutions to the problem of simulating single pieces of hardware. Hardware Description Languages (HDLs) support the simulation of hardware on a low layer of abstraction. The hardware descriptions used in this kind of simulation typically enable the manufacturing of the hardware itself. This means that the description, and thus the simulation, of the device is very accurate. This accuracy comes with the cost of low performance. To simulate more complex systems, a higher layer of abstraction is needed. This increases the simulation speed but decreases the resulting accuracy. A HDL that supports multiple layers of abstraction, such as SystemC [1], can describe the interconnection of hardware components at a high layer of abstraction and, at the same time, keep most of the accuracy for the components themselves [2].

The simulation of a network of computing devices can be challenging. Most of the simulators available are either limited to the sole simulation of the interaction between the network components (e.g., [3]), or can just give rough estimations of the executing time on each component (e.g., [4]).

To be able to simulate a sensor network without sacrificing the accuracy requires the connection of a hardware simulation tool and some tool that performs the interaction between the sensor nodes. Pieber et al. [5][6] described an approach to connect one instance of a SystemC simulation to the Gazebo simulator, a tool that is able to simulate an environment for the sensor. We want to extend this approach by instantiating multiple sensors in the Gazebo simulation and connecting them in this environment.

To test the resulting performance change, a small-scale networked control system has been implemented. This system is then performing some data acquisition, forwarding the data through another network node and finally the data storage at a final sensor node. In contrast to the simulation created by Pieber et al. [5] where one SystemC instance is connected to the Gazebo simulation, this simulation connects multiple instances of SystemC simulations to the Gazebo environment.

The remainder of this paper describes this process. Section II briefly describes the background information and states the related work to this publication. In Section III, the design of our approach is described. The implementation of the necessary parts is described in Section IV. Section V highlights our findings. This publication concludes in Section VI. This section also states our thoughts of what can be done next.

II. BACKGROUND AND RELATED WORK

As concurrent tasks are a vital part of any simulation, mechanisms to cope with this are introduced in every modelling language. There are three main approaches to deal with concurrency in simulations:

1) **Multi-instance-one-simulation** One simulation contains multiple instances of parallel tasks.
2) **One-instance-multi-simulation** One model only contains a single task. These tasks are then run in multiple simulations that communicate with each other.
3) **Multi-instance-multi-simulation** Each simulation contains multiple concurrent tasks. The tasks communicate within a simulation, the simulations communicate within themselves.

A simulation written in SystemC and Gazebo are simulations of the first type. SystemC simulations run its component modules quasi parallel using delta-cycles to reach a stable state. Then, the simulation time is advanced until the next triggered event happens. In the Gazebo simulator, the components are modeled via plugins. These plugins are called sequentially. When all components have been executed, the simulation time is advanced by one time increment and the process restarts.

A more powerful example of a SystemC simulation comprising of multiple instances of modules has been created by

 ISBN: 978-1-61208-744-3

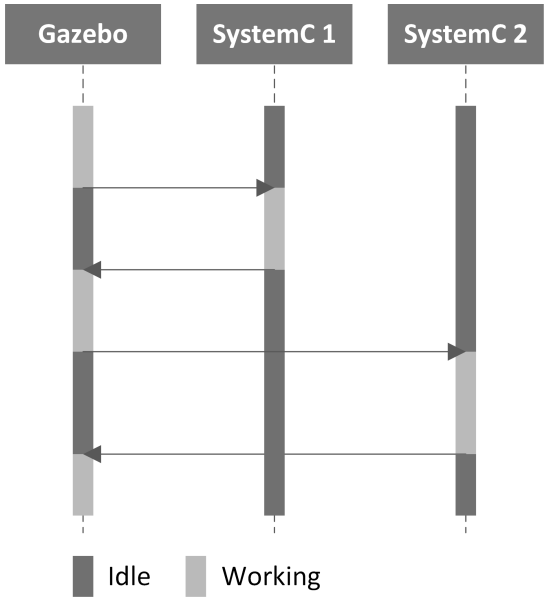

Figure 1. Performance problem using Gazebo and multiple SystemC instances.

Park et al. [7]. In this simulation, a smart house was simulated using SystemC. There, each electrical component has been modeled as a part of the complete simulation. These models are evaluated in parallel.

The combination of these two simulations results in a concurrent simulation of the third type. This is a simulation with multiple instances of different simulations, each containing multiple concurrent modules.

The process Pieber et al. [6] described in their work is intended to simulate one sensor node in an environment. This environment provides the sensor with stimuli that the sensor can work with. Furthermore, a communication channel is provided to communicate with the sensor and get feedback during the simulation time. In this approach, the Gazebo simulation is halted during the execution of one SystemC simulation. This results in a performance issue when using multiple SystemC instances. Figure 1 illustrates this. Here, one SystemC simulation blocks all other simulations. This results in an execution pattern suitable for a single processor core where only one simulation is being executed at a time. In this paper, we try to parallelize the SystemC simulations by introducing a mechanism that starts all SystemC simulation steps in parallel and waits for all to finish.

There are some proposed solutions to parallelize SystemC, such as [8]-[11]. These describe approaches that look for all executable SystemC modules and try to execute them in parallel. This results in a Type 2 concurrent simulation (One-instance-multi-simulation) as each executable model is treated as a single simulation. In these approaches, one large simulation is split into multiple concurrent simulations that are spread over the cores of one computer. In contrast to this, the approach described here uses multiple SystemC simulations and distributes them via a network. This method can spread the simulation on cores of the same machine, but also use additional computational resources of other computers in the network.

Schumacher et al. [9] presented an approach to simulate an Multi-Processor System on Chip (MPSoC). As their solution uses threads to achieve the parallelism, the solution is tied to a single host machine. While the authors claimed a significant performance improvement, it is already argued that this approach is not sufficient for a large sensor network simulation.

The approach of Sinha et al. [11] splits one SystemC simulation into multiple executable processes. In this approach, not only multiple cores or the Central Processing Unit (CPU) of one computer can be utilized, but also the Graphics Processing Unit (GPU).

Chopard et al. [12] propose a method to parallelize the SystemC kernel. In their approach, the researchers achieve a speedup comparable to the number of usable CPU cores.

The article of Jones [13] describes a more optimistic approach of parallelizing SystemC simulations. Jones also addresses the topic of race conditions that can occur when running such simulations in parallel. He also mentions that his technique of accelerating SystemC simulations is not suitable for existing simulations as large portions of code would need to be modified.

The possibility to accelerate SystemC simulations that rely on discrete events is discussed by Dömer et al. [14]. This team of researchers present a scheduler that spreads the runnable simulation nodes on the available CPUs.

Huang et al. [15] presented a SystemC library to handle the distribution of SystemC simulations. This approach is suited to work for multi-core machines as well as for a number of separate hosts. A downside of this approach is the limitation to functional and Transaction Level Modelling (TLM) simulations. Another disadvantage of this approach is the need for every SystemC simulation to handle its own communication with the rest of the simulation. Both of these issues are reflected on in our work by enabling SystemC simulations to be independent of the distribution architecture.

Another approach for simulating multiple computers in a network is used by Simics [3]. This simulator is built such that it can use multiple computers in a network to simulate the interaction of the systems. In this approach, the simulated network nodes are connected via a simulated network. This simulation can be used to simulate a computer network at a high level of abstraction. The integration of analogue signals (measurements of a sensor, or a very low level of abstraction) is not directly possible.

Clement et al. [16] coined the term Internet of Simulation. In their paper, the authors describe the need for heterogeneous simulation systems that can capture the complex nature of Cyber-Physical Systems (CPS). In their terminology, this paper describes a co-simulation for virtual engineering.

This paper is based on the publications of Pieber et al. [5][6]. It improves in the following details:

- **Concurrency:** The original approach uses one Gazebo plugin for each sensor in the environment. Each of these plugins directly creates a SystemC process and communicates with it. This entails that

all SystemC processes are located on the same host machine. With the approach presented here, the SystemC instances are started before the main simulation. These simulation instances can be located on different host machines and communicate via a network to the main communication. The plugins in the main simulation communicate to a server plugin that handles the network traffic and connects the SystemC simulations to the intended plugins.

- **Modularity:** As the SystemC simulations are started before the main simulation, multiple different implementations of the same simulation can be connected to the same sensor plugin. This increases the modularity of the simulation as only few changes need to be made in order to exchange the SystemC simulations.

III. DESIGN

To improve the design of Pieber et al. [6] we implemented a server-client structure to spread the simulations to multiple computers. Using this, only a single plugin (the server) connects the SystemC instances. This plugin then blocks the Gazebo simulation until all SystemC tasks are finished. Figure 2 shows the intended execution path. The Gazebo simulator performs the calculations that are necessary for the data transmission. The SystemC simulations receive the data and perform operations on the data. When the simulation steps of all SystemC instances are finished the Gazebo simulator can continue its operation.

In this design, all necessary data from Gazebo is generated during its time step. This information is gathered in the server plugin. The server plugin then forwards the information to the SystemC instances. While the SystemC simulations are performing the simulation step, the Gazebo simulation is halted. During the execution of the SystemC simulation, the generated data is transmitted to the Gazebo simulation. There the server plugin captures the data. When the simulation step is finished, the SystemC simulations are halted and the Gazebo simulation can continue. In this way, no information is lost between the simulations, and all simulations are synchronized at the end of the time steps. As the server starts all SystemC simulations and waits for all to finish, also the SystemC simulations are synchronized.

In this design, the server does not contain simulation relevant operations. It just connects the Gazebo representations of the sensors (sensor plugins) to the SystemC simulations. This server therefore acts as a gateway for the sensor plugins to the SystemC simulations which can be executed anywhere in the network.
The server plugin has three responsibilities:

- **Connecting** the correct SystemC simulation to the intended sensor plugin. This includes the identification of the connected SystemC simulations and the matching to the correct sensor plugin.
- **Passing data** between the sensor plugin and SystemC simulation.
- **Synchronizing** the simulations.

The server plugin runs a thread that listens for incoming connections from remote SystemC simulations. For each

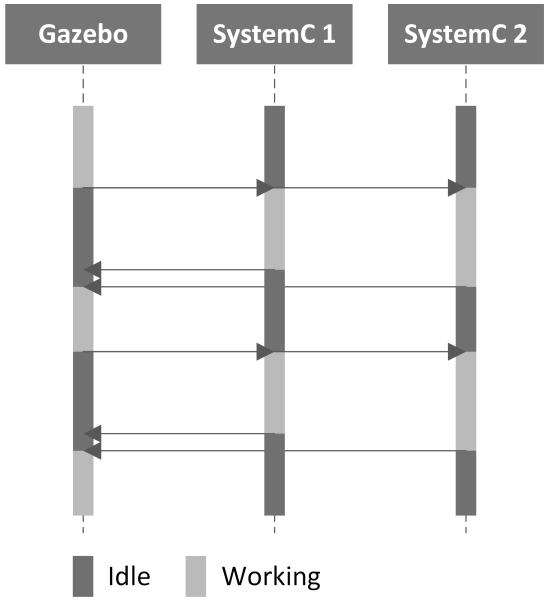

Figure 2. Improved handling scheme for multiple SystemC instances.

connecting simulation, a thread is created that handles the information exchange with the sensor plugin. Additionally, this thread forwards the information to the SystemC simulation and receives the resulting information. Figure 3 shows the top-level structure of the plugin server. The server listens for new SystemC connections and creates a worker thread for each connected simulation. Each worker thread manages the communication between the sensor plugin and the SystemC simulation. To handle the SystemC simulation and the synchronization of the simulations, the thread appends data about simulation states. In addition to all of that, the thread keeps information about the connection to the SystemC simulation. This is information about the Internet Protocol (IP)-address, the port number, the simulation identifier, and the last sent command to which the SystemC simulation has to react.

Each worker thread starts by initializing its own memory. This is followed by the initial checks of the SystemC simulation. These checks are performed by verifying an identifier. If the SystemC simulation can be used for the plugin, an initial configuration for the SystemC simulation is sent. When the initialization is finished, the SystemC simulation needs to respond with its state. For each simulation step, the data for the SystemC simulation is gathered by the sensor plugin. This data includes the change of sensor data, information about incoming messages, and changes of external energy sources (for energy harvesting). This gathered information is then forwarded to the server plugin and the correct worker thread. The worker thread packs the data and adds additional information. This additional information includes changes of simulation states, status information, or commands to the interface on the SystemC side. When the server plugin is being executed, all information is forwarded to the SystemC simulations by the worker threads. Until all worker threads have received the signal that their SystemC simulation has finished its execution, the Gazebo simulation is blocked. During the execution of the SystemC simulation, data that is destined for the Gazebo simulation is sent to the worker thread. This

 ISBN: 978-1-61208-744-3

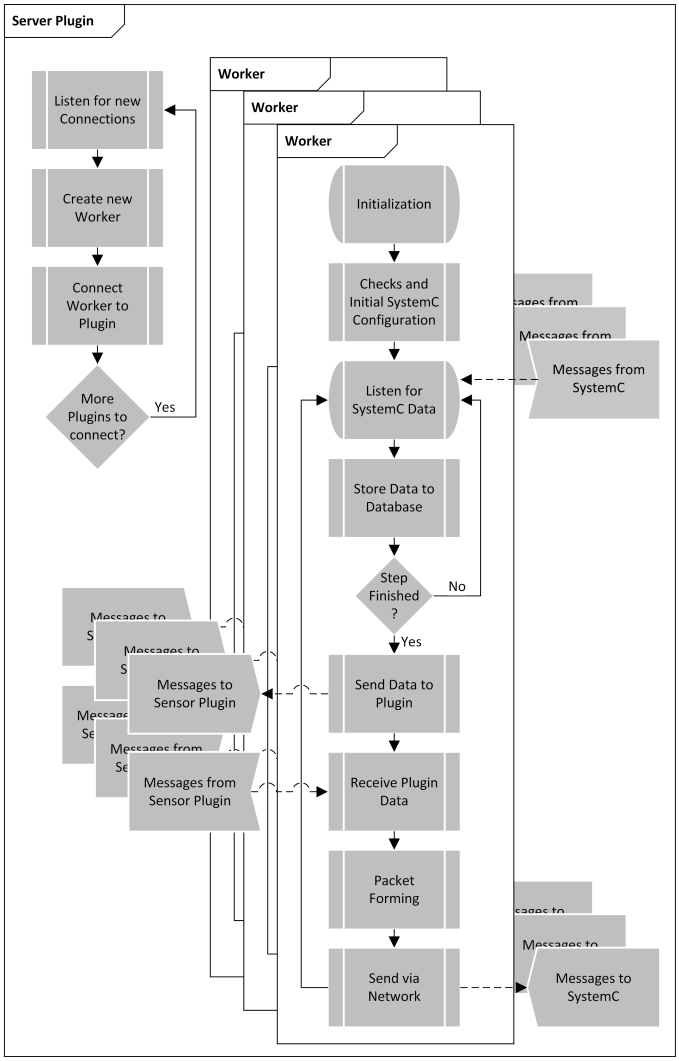

Figure 3. Top-level structure of the server plugin.

information is stored until the simulation proceeds. At the end of the SystemC simulation step, a signal is sent to the Gazebo simulation, informing the worker about the simulation status. If all SystemC simulations have stopped their execution, the Gazebo simulation can proceed. Now the stored data is transferred to the sensor plugins. The plugins can access the data in the next Gazebo time step.

With the use of the parallel design, the computation of the network should have a similar performance as the sole computation of the slowest node in the network. Thus, when simulating similar nodes, the performance gain should be related to the number of parallel nodes.

To test this hypothesis, multiple tests are designed:

1) **All nodes the same - single** This test simulates one sensor node performing measurements.
2) **All nodes the same - sequential** This test comprises of three identical nodes, each performing the same measurements.
3) **All nodes the same - parallel** The three nodes are run using the parallel computation design.
4) **Networked system - single** In this test each of three different nodes is simulated on its own. The data for nodes two and three is computed by node one and given as input for the simulation.
5) **Networked system - sequential** This tests simulates all three nodes in the network using the sequential approach. The data is generated and encrypted in node one, transmitted over node two, and decrypted and stored in node three.
6) **Networked system - parallel** The three nodes of the networked system are simulated using the parallel simulation design.

The first three tests act as a baseline test for the hypothesis that the simulation performance of the complete simulation is comparable to the simulation of the slowest node.

Test 1 simulates each node separately. As the three nodes are identical the simulation time should be equal as well. Test 2 uses the old connection via Gazebo to test the simulation speed of the sequential case. Each SystemC simulation is identical. As they are executed sequentially, the simulation time should be roughly the sum of the single simulations - in this time three times the duration of Test 1. Test 3 uses the new parallel design to connect the sensors. As all simulations are calculated in parallel, the execution time should be similar to the simulation of one node of Test 1.

The second three tests use three different nodes. One that gathers data, one that transmits the data to the last one, and one that receives and stores the data. The data is continuously transmitted and sent across the nodes. This way all nodes are active all the time (except for the first message). Otherwise, the test would limit the execution to the sequential case.

Test 4 uses precomputed data to test the functionality of each node individually. Each node is simulated separately. Test 5 combines the three nodes using the old connection via Gazebo to test the sequential case. Test 6 uses the new parallel design to connect the nodes.

The resulting simulation represents the data flow between the sensor nodes on the environment scale. Furthermore, as the sensor nodes themselves are simulated using SystemC, the data processing on each sensor node is being calculated.

IV. Implementation

The most notable change to the design of Pieber et al. [6] is that a new plugin has been implemented that performs the communication tasks. In their publication, every sensor plugin communicates with the corresponding SystemC instance. We have implemented another plugin that performs the communication tasks for all sensor plugins. A concept for our implementation can be seen in Figure 4. There, the *sensor plugins* send their data to the *plugin server*. This server handles the communication with all *SystemC* instances. When the SystemC simulations have returned data, the *plugin server* forwards the information to the *sensor plugins* in the next time step.

The communication between the server and the SystemC clients is based on the concept described by Pieber et al. [6]. An additional top-level structure is added to the EXtensible

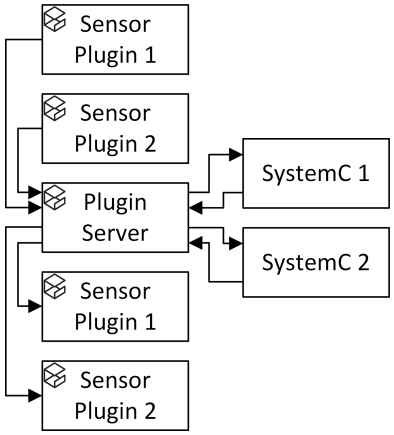

Figure 4. Concept for transmitting data between the sensor plugins, the plugin server, and the SystemC simulations.

Markup Language (XML)-formatted data. This structure specifies the type of message and the according payload. There are two different types of messages that can be sent between the server and client:

1) **Command:** This type specifies a message from the server that the simulation needs as input data. This can be a sensor value, incoming messages, or instructions to change the simulation status (finish the simulation, change the simulation step size).
2) **Status:** The status message can be sent from either side of the communication. If it is sent from the server, it can ask for the identification of the SystemC simulation, request information about the simulation status, or inform the SystemC simulation about its own simulation status. A status message from the client side can contain requested information, or signal the server that the simulation step is finished.

The complete communication structure between the server and a client can be seen in Figure 5. The Gazebo simulation starts the server. The server blocks the simulation until all SystemC clients are connected. After that, the server receives the commands to send from the sensor plugins. This information is relayed to the appropriate client. With this message the client is given the command to start the simulation step. Until all SystemC simulations have finished their step, the server blocks the Gazebo simulation. When all clients are ready, the Gazebo simulation can be resumed. This results in another message to the SystemC client. If the Gazebo simulation is to be ended, the server disconnects from the SystemC client. This triggers the reset of the SystemC simulation. As the simulation system is built to run in a network, simulation clients other than the intended ones could connect to the server. To enable the server to check if the SystemC simulation is required in the current Gazebo simulation run, an ID is requested from the client. This ID specifies the type of simulation. Based on this, the server can decide whether the client should be accepted or rejected. Accepted clients are then logically connected to the appropriate sensor plugin.

During the execution of the SystemC tasks, the plugin server blocks the Gazebo simulation. When all SystemC instances have returned their signal that the simulation step has been finished, the plugin server resumes by distributing the received information to the sensor plugins.

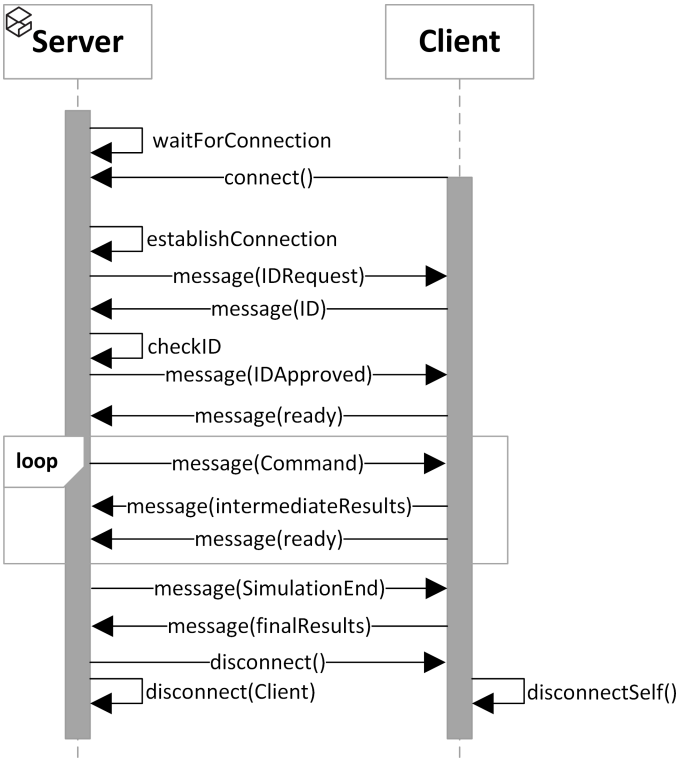

Figure 5. Messages between the server and one SystemC client.

TABLE I. Durations of the test simulations

Test Nr.	Node Nr.	Duration
1	1,2,3	$\sim$ 15.34 sec
2	1,2,3	48.0487 sec
3	1,2,3	16.7513 sec
4	1	153.7706 sec
4	2	10.6859 sec
4	3	120.8714 sec
5	1,2,3	284.6381 sec
6	1,2,3	156.1416 sec

V. RESULTS

The results of the experiments introduced in Section III are listed in Table I.

These six experiments show that the final execution time is slightly larger than the longest component simulation. It furthermore shows that the simulation time, compared to the sequential case, can be multiplied with a factor of $\sim \frac{1}{N}$ where N is the number of parallel SystemC simulations where each simulation uses approximately the same amount of time. For simulations that differ in their simulation duration, the final duration is slightly longer than for the slowest simulation. This system therefore provides an extensible and flexible basis to add additional SystemC simulations. The Gazebo server remains independent of any added SystemC simulations while also the client side keeps individual SystemC models separated.

As an additional improvement can be seen that the sim-

ulations need not be calculated on the same computer as the Gazebo simulation. Normally, the Gazebo host system would need additional resources to run the Gazebo system and the SystemC simulation. The server-client structure allows the SystemC simulations to be spread over a network. Thus, the amount of possible parallel simulations is not limited to the resources on one machine.

As a limitation to this system, the general network overhead should be mentioned. As the commands and data are sent over a network, additional data is added by the system. This adds to the data size that is to be sent. Furthermore, the data needs to be packed and unpacked at either side of the communication, increasing the latency. This should be considered, when designing the simulation.

VI. Conclusion and Future Work

This paper describes a method to connect multiple SystemC simulations of sensor nodes to a network. These sensor nodes are placed in a virtual environment, simulated with the Gazebo simulator, and communicate via this environment with each other. As the Gazebo simulator processes its components sequentially, the final simulation is inefficient. To improve the performance of this simulation, a server-client structure is proposed and implemented. This connects the server on the Gazebo side to the SystemC simulations via network sockets. The server can then unite the individual calls to the SystemC simulations and start all simulations in parallel. In contrast to the simulation duration being the sum of all component simulations, the duration of the simulation using this structure is only slightly longer than the longest component simulation.

In the current form, the simulation results can only be evaluated when the simulation is finished. Due to the lengthy simulations, this is inefficient. Therefore, live signal plotting can be implemented. Using this, the simulation operator can spot errors in the simulation early and stop the execution prematurely. This would then reduce the simulation time for erroneous runs significantly.

Acknowledgments

This project has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 692480. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Netherlands, Spain, Austria, Belgium, Slovakia.

IoSense is funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program "ICT of the Future" between May 2016 and April 2019. More information https://iktderzukunft.at/en/

References

[1] Accelera, "SystemC." http://accellera.org/downloads/standards/systemc, 2000. Last accessed on mar 18, 2019.

[2] P. R. Panda, "SystemC - A modelling platform supporting multiple design abstractions," in *Proceedings of the 14th international symposium on Systems synthesis - ISSS*, pp. 75–80, Association for Computing Machinery (ACM), 2001.

[3] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[4] V. Shnayder, M. Hempstead, B.-r. Chen, G. W. Allen, and M. Welsh, "Simulating the Power Consumption of Large-scale Sensor Network Applications," in *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, (New York, NY, USA), pp. 188–200, ACM, 2004.

[5] T. W. Pieber, T. Ulz, and C. Steger, "SystemC Test Case Generation with the Gazebo Simulator," in *Proceedings of the 7th International Conference on Simulation and Modeling Methodologies, Technologies and Applications - Volume 1: SIMULTECH,*, pp. 65–72, INSTICC, SciTePress, 2017.

[6] T. W. Pieber, T. Ulz, and C. Steger, "Using Gazebo to Generate Use Case Based Stimuli for SystemC," in *International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, pp. 241–256, Springer, 2017.

[7] S. Park, H. Kim, H. Moon, J. Heo, and S. Yoon, "Concurrent Simulation Platform for Energy-Aware Smart Metering Systems," *IEEE transactions on Consumer Electronics*, vol. 56, no. 3, pp. 1918–1926, 2010.

[8] p. Ezudheen, P. Chandran, J. Chandra, B. P. Simon, D. Ravi, *et al.*, "Parallelizing SystemC Kernel for Fast Hardware Simulation on SMP Machines," in *Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation*, pp. 80–87, IEEE Computer Society, 2009.

[9] C. Schumacher, R. Leupers, D. Petras, and A. Hoffmann, "parSC: synchronous parallel systemc simulation on multi-core host architectures," in *2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pp. 241–246, IEEE, 2010.

[10] A. Mello, I. Maia, A. Greiner, and F. Pecheux, "Parallel Simulation of SystemC TLM 2.0 Compliant MPSoC on SMP Workstations," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 606–609, European Design and Automation Association, 2010.

[11] R. Sinha, A. Prakash, and H. D. Patel, "Parallel simulation of mixed-abstraction SystemC models on GPUs and multicore CPUs," in *17th Asia and South Pacific Design Automation Conference*, pp. 455–460, IEEE, Jan. 2012.

[12] B. Chopard, P. Combes, and J. Zory, "A Conservative Approach to SystemC Parallelization," in *International conference on computational science*, pp. 653–660, Springer, 2006.

[13] S. Jones, "Optimistic Pparallelisation of SystemC," *Universite Joseph Fourier: MoSiG DEMIPS, Tech. Rep*, 2011.

[14] R. Dömer, W. Chen, X. Han, and A. Gerstlauer, "Multi-Core Parallel Simulation of System-Level Description Languages," in *Proceedings of the 16th Asia and South Pacific Design Automation Conference*, pp. 311–316, IEEE Press, 2011.

[15] K. Huang, I. Bacivarov, F. Hugelshofer, and L. Thiele, "Scalably distributed SystemC simulation for embedded applications," in *2008 International Symposium on Industrial Embedded Systems*, pp. 271–274, IEEE, 2008.

[16] S. Clement, D. W. McKee, R. Romano, J. Xu, J. Lopez, and D. Battersby, "The Internet of Simulation: Enabling Agile Model Based Systems Engineering for Cyber-Physical Systems," in *2017 12th System of Systems Engineering Conference (SoSE)*, pp. 1–6, IEEE, 2017.

 ISBN: 978-1-61208-744-3