Using Gazebo to Generate Use Case Based Stimuli for SystemC

Thomas W. Pieber, Thomas Ulz, and Christian Steger

Graz University of Technology - Institute for Technical Informatics, Graz, Austria
{thomas.pieber, thomas.ulz, steger}@tugraz.at

Abstract. Realistic simulations of new hardware are of utmost importance to achieve good results. The current approach to such simulations is that the Device under Test is exposed to stimuli that are either generated randomly, or that are generated by engineers reverse engineering the use cases and extending the inputs by some extreme cases. In this paper we describe an approach to generate useful stimuli for a SystemC simulation directly from a simulation of the use case. In this approach the use case is simulated using the Gazebo simulator. The stimuli for the Device under Test are then extracted and sent to the SystemC simulation.

1 Introduction

In the development of new systems simulations need to be performed to find errors early. With such simulations the developed system (or Device Under Test "DUT") can be tested extensively and optimizations and error corrections can be implemented quickly and inexpensively. These simulations are usually stimulated with events that occur in the expected use case as well as some extreme cases. These tests are designed by engineers reworking the scenarios and defining the inputs and the expected behaviour. In addition to these tests, random input sequences can be applied to test the DUT's reactions to faulty or unexpected inputs as random input is unlikely to be valid. All together that means that the current test procedure consists of valid inputs designed by the system engineers and (mostly) invalid inputs generated by random testing. We therefore propose an architecture for a generator that can produce valid inputs to the DUT design which can also be evaluated according to the expectations of the engineers. Such system can decrease the effort needed to design tests for the DUT, as only the valid scenarios need to be described. These will then automatically generate valid input data and the expected output.

For such simulation the environment in which the DUT should operate can be simulated. This environmental description only needs to describe the essential parameters that can affect the DUT. Such simulations can be performed in a simulator such as the Gazebo simulator [6, 13]. This simulator is designed for robotic use cases and is designed to handle complex systems and generate accurate sensor information of any kind. This open source simulator also allows modifications to be as useful and accurate as we want it to be. These modifications are done by implementing plugins for the environment (world), the models,

the sensors, the simulation core, the visuals, and the GUI. This simulator operates in discrete time steps of 1 ms. This degree of simulation accuracy is enough to simulate the movement of robots and sparse enough that the robot's operating system can handle most commands in this time step.

To simulate our DUT another tool such as SystemC [1] can be used. With this tool a complex microsystem can be designed and tested. Furthermore, the component parts of the system can be modelled in various degrees of detail. This allows for accurate simulations or even synthetization of the newly developed parts and efficient simulation of existing hardware. SystemC operates in discrete time intervals as small as 1 fs.

When combining these two simulations, this difference in simulation speed poses a major problem. The execution of a test scenario can last for many minutes. In combination with the fine grained simulation time steps of SystemC this can generate huge amounts of data which need to be handled. This problem needs to be considered when choosing the traced signals and information that should be transferred between the simulations. Additionally, the testbench of the SystemC simulation must be altered to include the communication between the simulations.
This leads to the issue of the communication itself. The simulations need to exchange data such as the generated input and output of the simulation step, as well as status information about the simulations itself. The difference in time steps also introduces the problem that SystemC requires data in more detail from the Gazebo simulation. This data is to be estimated and extrapolated from the existing inputs. It also produces output data that is filtered to allow Gazebo to work with the resulting data.

This paper is based on the work done by Pieber et al. [15]. It expands the ideas behind that publication, gives more detail on the design and implementation of the combination of the simulations. It furthermore expands the evaluation by constructing a detailed simulation run and interpreting the results.

The remainder of this paper is structured as follows: In Section 2 other works that combine SystemC or Gazebo with other simulators are described. Section 3 explains the motivation for our design, states the requirements that need to be implemented, and gives details on the solution for the requirements. An evaluation of the design is described in Section 4. This is done by constructing and analyzing a sample simulation run. Following that, Section 5 mentions ideas on how to further improve the proposed design. This paper concludes with Section 6.

2 Related Work

As Gazebo is an open source simulator for robotics it is primarily combined with a robot operating system such as ROS [19] or YARP [9] to control the simulated robots [10].
There are approaches to combine the Gazebo simulator to software for robotics and computational intelligence [21]. There are further works that connect other

tools that can simulate hardware [8], but the main approach in these works is to use the interface from Gazebo to ROS and implement ROS nodes to connect to the rest.

A design process for SystemC is given by Panda [14]. With this language a model for complex systems can be described and executed. There are many publications that implement interfaces to SystemC as it provides a good basis for simulations [3, 4, 11, 16]. In these approaches the functionality of SystemC is extended to provide the functions needed by the researchers.
An interface from SystemC to Matlab/Simulink was designed by Bouchhima et al. Here the SystemC simulation was stimulated by a continuous environment simulation written in Matlab/Simulink. SystemC was also connected to analog circuit simulators like SPICE [5] and VHDL [2] to improve on flexibility and simulation performance.
Mueller-Gritschneder et al. developed a robot simulation platform in SystemC [11]. They simulate the behavior of the robot on the transaction layer and forward the results to an environment simulation written in Java. They do this in order to simulate the movement of the robot as accurate as possible. In this paper the robot is simulated in SystemC, while in this proposal the robot's behavior is the input to the SystemC simulation to simulate parts of the environment.

In summary SystemC was connected to many other simulations. It is then used as core for other simulations or to generate more accurate results. In the context of robotics, SystemC has been used to simulate the movement of the robot. In contrast to that, this publication uses the robotic simulation to stimulate SystemC components with inputs from the environment to automatically generate valid stimuli.

In this paper, a use-case is evaluated where a sensor measures data from the environment, and is read out and charged via Near Field Communication (NFC) by a robot. Some publications describe the techniques used to transmit data alongside energy and storing the excess energy in small batteries or capacitors [7, 17, 20].

To connect simulations a common interface must be created over which data can be exchanged. In this approach the common interface used is a POSIX (Portable Operating System Interface) pipe where XML (Extensible Markup Language) formatted data is sent. Another possibility to format data efficiently is the JSON (JavaScript Object Notation) format. This would be more efficient than XML [12], but due to other reasons, explained below, the XML format is chosen. Gazebo uses Googles Protocol Buffer (ProtoBuf) as formatting method to transmit data internally. Sumaray and Makki compare the efficiency of this protocol to XML and JSON [18].

Based on [15], this publication expands on the detail of the design and implementation of the developed connection of the simulations. Main focus of the expansion is on details concerning the Gazebo plugin. Additionally the evaluation of the system is expanded. Here, detailed information on the traces produced by SystemC is given.

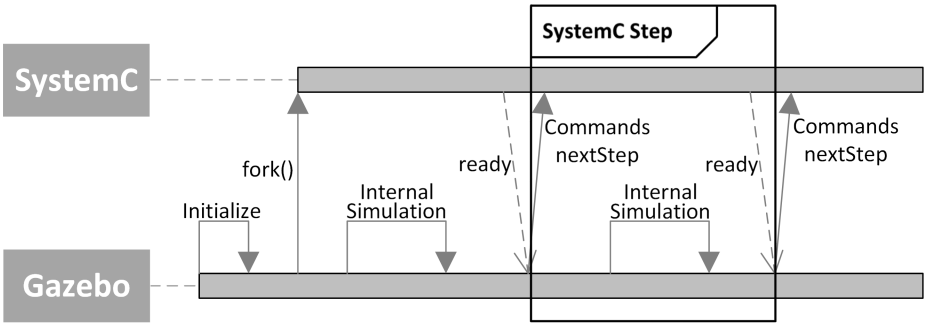

Fig. 1. States of the execution of the implemented plugin.

3 Design and Implementation

The goal of the presented design is to find a method to generate stimuli for a SystemC simulation automatically and being able to see how the simulated system behaves in the specific use-case. To enable this, a connection between SystemC and a high-level simulation is established. This high-level simulation (in this approach the Gazebo simulator) represents the surroundings of the newly developed system (a sensor in this use-case). Using the data from the Gazebo simulator, stimuli for the sensor can be created. That means that the environment of the sensor becomes the de-facto testbed. With this method the stimuli for the SystemC simulation are generated by the interaction of the sensor with the environment. This generates the stimuli not only faster than an engineer could, but also only small variations in the environment can generate a wide variety of different test scenarios such as more noise in the communication or energy fluctuations due to movements of the reader or changed mutual inductance due to small changes in the distance between the antennas.

To connect two simulations successful, both must support the interfaces necessary. To do so, an overall structure for the communication was developed. This structure is shown in Figure 1. This plan visualizes how the simulations are connected, how they communicate and when operations are performed. This figure also shows the minimum requirements that this approach needs to work. In this example the SystemC process is forked from the Gazebo simulation. Here some initial configuration concerning the communication can be set up. One step of the Gazebo simulation then invokes one from SystemC. A return message from SystemC informs Gazebo that the simulation step of SystemC is properly executed. During the execution of the SystemC step additional messages for Gazebo may be sent that need to be captured from the Gazebo environment.

Gazebo can be extended by the use of plugins. To apply the input of the Gazebo environment to a SystemC simulation, one such plugin that handles the communication needs to be developed. The structure for that plugin can be seen in Figure 2. This structure implements the following five operations.

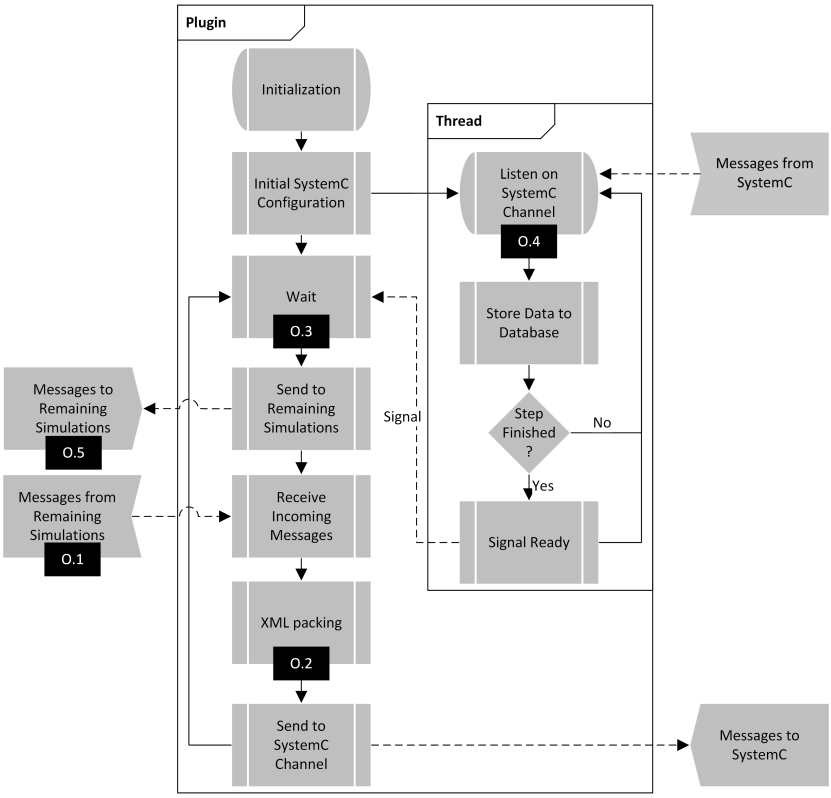

Fig. 2. States of the execution of the implemented plugin.

O.1 The required data must be collected from the environment. That includes data that the sensor can measure as well as communication data.
O.2 The collected data needs to be packed into messages that can be sent to the SystemC simulation.
O.3 The plugin needs to halt the simulation of the environment until the SystemC simulation step finishes.
O.4 During the execution of the SystemC step, all messages needed for the remaining simulation(s) need to be received, stored, and ordered.
O.5 The collected information needs to be distributed to the remaining simulation(s). This can be communication data, visual data, or status information.

The operation defined in O.1 is needed to generate valid input to the sensor that can be evaluated. To generate this information a world plugin may be needed to gather the information. This operation can then be achieved by generating a Gazebo internal communication from this plugin to the plugin connecting the SystemC simulation. Additionally a communication path between the communicating entities must be established. This communication may also be altered in order to simulate the effect of the channel.

To send the gathered information to SystemC operation O.2 is required. This operation formats the data in a way suitable for transport to SystemC. To send arbitrary data the data items are converted into string format and packed using an XML structure.
To properly synchronize the two simulations Gazebo needs to be able to wait for SystemC to finish calculating the current time step. That implies that the plugin needs to be able to halt the Gazebo simulation until it receives the signal indicating the completion. This operation is referred to as O.3.
While Gazebo is waiting for SystemC to finish, SystemC may send different messages that are needed for the rest of the Gazebo simulation. This information needs to be processed and stored until Gazebo can simulate the next time step. When this happens, the plugin needs to forward the information received from SystemC to the rest of the simulation. This operation is described in O.4.
Operation O.5 describes the correct distribution of the gathered data. As SystemC can send different data (e.g.: visual updates such as LEDs, or data for communicating with other entities), the information needs to be split into the topics and sent to their destination using an internal communication mechanism.

The difference in simulation speeds is one of the biggest hurdles in connecting the two simulations. As a tool for simulating interactions and movements of robots, Gazebo works with time steps of 1 ms. On the other hand, SystemC can handle steps as small as 1 fs. This is needed for simulating hardware components as also a "slow" computer which only works with 50 MHz performs $5 \cdot 10^4$ operations in one time step of Gazebo. This difference of twelve orders of magnitude of the simulations can result in massive amounts of data generated by SystemC which is hard to evaluate in Gazebo. Therefore, some measures to limit the amounts of data that are transferred need to be implemented. This can be done best when defining the requirements for the connection between the simulators.

As the two simulations must be compatible in their interfaces to each other, the structure of a SystemC simulation needs to be adopted. Figure 3 shows the overall structure of such SystemC simulation. The messages coming from the Gazebo simulator are received and analyzed. If some parameters need changing, the adaptations are done. To reduce the simulation time, it is evaluated if the changes require instant action. Should that not be the case, the time that should be simulated is added as a debt in comparison to the Gazebo simulation. An action is required if the time debt is too large or if the sensor receives messages that require an answer. To simulate these actions in the correct order, the old parameters and commands are reset and the simulation is run to balance the time debt. In this run the conditions at the current time instance are estimated. After that the new parameters are set and the simulation is started for this time step. When a stable condition is reached the simulation is halted and the conditions for the end of the step is calculated. Should the calculations need more time than one time step, the checking if action is required evaluates to "yes" in the next time step.

To send arbitrary messages between the two simulation environments, an easy-to-implement approach is used. As the SystemC simulation gets forked from

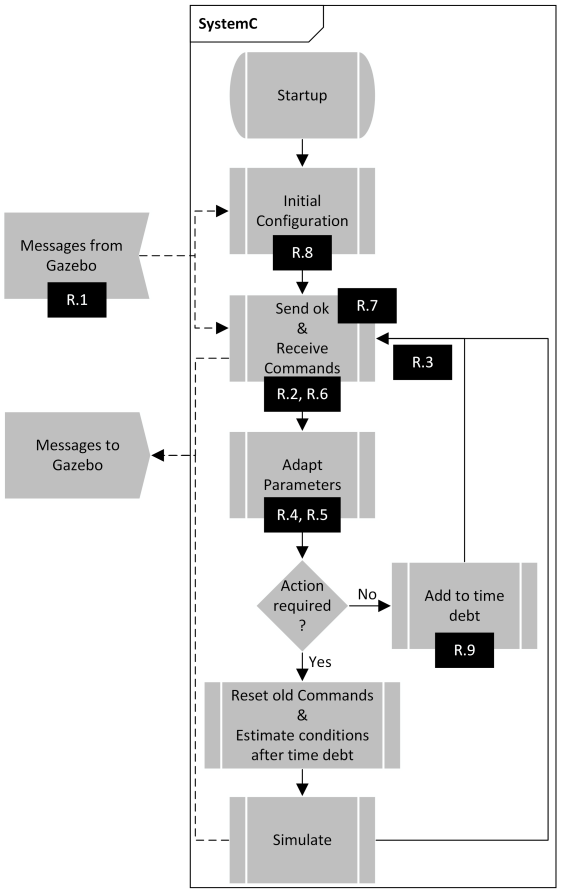

Fig. 3. Structure of the SystemC simulation.

the Gazebo plugin (Figure 1) the standard input and output can be redirected to a POSIX pipe. The Gazebo plugin uses that pipe to exchange messages with the SystemC simulation. This interface allows the transmission of string-type messages. So the commands and parameters need to be packed in a format that can be sent as string and the string received needs to be parsed in order to get the information back. An easy to implement method is the use of XML (Extensible Markup Language) data structure to pack the information in strings. That means that the interesting values are encased in XML-style tags. With these tags the string can be split in parts containing different types of data, which are then evaluated. The use of XML also allows the SystemC components themselves to send their information as soon as it is available and Gazebo can receive and process the data in the order it was produced. Although JSON would be more efficient than XML, XML was chosen because this makes the composing of messages inside SystemC more easy. With XML the messages can be sent

whenever the information is available, with JSON on the other hand the Information needs to be packed in smaller JSON objects which can be sent. This can introduce additional computational overhead, leading to JSON loose its better performance.
The data sent by the SystemC simulation is received by the Gazebo plugin and parsed. The incoming data stream is split into data chunks according to the XML-tags, preprocessed and stored in a fitting datastructure. When the SystemC simulation halts for the time step, a tag for synchronizing the simulations is sent to Gazebo. The reception of this tag signalizes the Gazebo plugin to transmit the collected data to the rest of the simulation and resume its work.
Gazebo is capable of using many types of communication, but Google's Protocol Buffer (Protobuf) is the most efficient communication it supports [18]. To fully use this method, custom messages can be defined that can hold various types of data. The messages from the sensor that are intended for the robot (in this scenario) are for example transmitted to a simulation of the transmission channel. There environmental information is used to simulate signal degradation due to free space loss and multi-path interference. The channel is simulated separately from the rest to be able to test different communication channels such as WiFi, Bluetooth, Zigbee, or in this case NFC. The calculations of the transmission statistics are based on [7, 20]. The channel then modifies the transmitted data and forwards it to the robot, where it can be accessed.
Communication from the robot to the sensor follows the same rules. The robot sends the data to the channel simulation. There, errors are introduced and the signal strength and received energy is calculated. This information is then sent to the sensor plugin which also collects the data for the sensor system itself. The collected data is packed in an XML message and sent to the SystemC simulation. The SystemC simulation receives the commands, parses the XML data and performs the actions needed to simulate the sensor accurately. To accomplish this, the original testbed must be modified to accommodate the interface to the Gazebo simulation. In order for that to work properly, the following requirements need to be fulfilled:

R.1 Gazebo must be able to transfer information about the simulation, commands and parameters for the DUT to SystemC.
R.2 To be able to react to changes in the environment, the SystemC simulation needs to operate in steps. Between the steps the information exchange can occur.
R.3 The SystemC interface for the communication must be able to parse and distribute the received information.
R.4 To support different types of simulation, the simulation time step of SystemC should be variable at each step.
R.5 Parts of the SystemC simulation that need special information need to be able to get it directly.
R.6 Commands received from the Gazebo simulator need to be executed in the order they arrive.
R.7 The two simulations need to be synchronized during the execution.

R.8 To reduce the memory required to run the simulation for extended periods, traces from the simulation should be able to be deactivated.
(R.9 The simulation should be done as quickly as possible, while maintaining the accuracy where needed.)

In addition to the changes in the testbed - now the interface to the Gazebo simulation - some adaptations in the rest of the simulation have been made. The most notable adaptation is the insertion of messages that are sent to the Gazebo simulator. These changes need to be made as the bulk of information is evaluated during the simulation by the Gazebo simulator instead of after the simulation. These changes are made in a similar fashion as the changes needed for requirement R.7.
Figure 3 additionally shows the parts of the SystemC simulation where the implementations of the requirements are mostly located.

To be able to fulfill requirement R.1 a POSIX pipe can be used. This allows the transport of information in string format between SystemC and Gazebo. For that, the standard input and output of the SystemC simulation are rerouted.
An XML parser is needed to split the gathered information into the data chunks needed for the different settings and commands. With the use of a global datastructure the distribution to all parts of the simulation can be performed. These measures fulfill requirement R.3.
The call of the start procedure for the SystemC simulation ("sc_start(...)") is inside a loop. The loop is halted when the simulation waits for input from the Gazebo simulator and the condition to exit the loop is sent from Gazebo when exiting or resetting the Gazebo simulation.
To allow a multitude of simulations the time step size of SystemC may need to be changed during runtime. This is represented in requirement R.4. To meet this requirement two special commands are added. One to set the time unit and one to set the numerical value of the time step.
To support requirement R.5 we modified not only the testbed, but also parts of the simulation. For every module we want a direct communication path to, we need a special tag to extract the data. This data is then written into a global datastructure to be accessible by all modules. The interface from the module itself retrieves the data from this datastructure and can perform the needed operations without the need to communicate the information through the layers of the module. The finished data is then stored in FIFO (fist in, first out) structures to wait for the simulation to need it. This requirement is especially useful for this scenario as we modeled a sensor system and the data the sensor measures is represented in the Gazebo simulation.
Requirement R.6 can be met with the same strategy. This time the FIFO buffer storage is located in the control unit of the sensor. Now the commands that have arrived can then be loaded and executed in the order they should have arrived. To properly execute the commands an additional field that stores the time when the command should have arrived is needed to simulate a serialized channel.
To get a proper time synchronization (R.7) the SystemC simulation needs to send a marker message back to Gazebo indicating that the step has been pro-

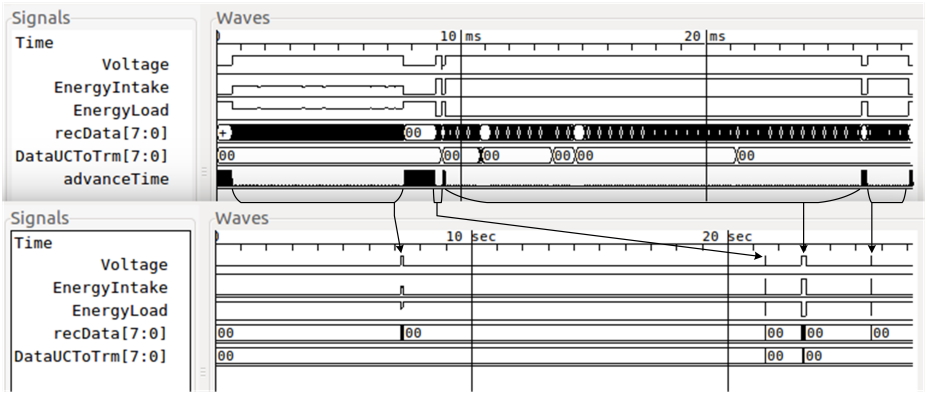

Fig. 4. Compression of idle-time.

cessed. This is done by declaring an extra XML-tag. After receiving this signal the Gazebo simulation is allowed to execute another step - again triggering the SystemC simulation.
To use a simulated environment as generator for system stimuli it is necessary to simulate longer periods of time before the system is initially triggered. This is done in order to generate different edge conditions on the simulated device. As we do not know what exactly the sensor measures during this time, it is necessary to also simulate this time in SystemC. Furthermore, the environment simulations can be run for an extended period of time between stimuli. When storing all generated data large files are created. Requirement R.8 refers to that problem. This can be solved by activating only traces that are needed. The information which traces are needed can be received during the initialization process. This information does not only contain which traces are needed, but also if the traces should be active at all, and what the file name should be.
To further decrease the need for memory, a detector system is be implemented that determines weather a simulation step can be stopped prematurely, or even needs to be started. This detector has the potential to reduce the simulation time significantly and corresponds to the optional requirement R.9.

Such detector can only be implemented if detailed knowledge of the inner workings of the simulation and the system it simulates is available. This also requires some major appendices to the existing simulation.

When pausing the simulation prematurely, two challenges emerge. The first one is the desynchronization of the two simulations. As SystemC offers no methods to change the simulation time when it is stopped, we introduced flag signals that get triggered if the simulation time should get changed. With the help of these signals and some post processing of the generated traces the synchronization can be restored afterwards. Figure 4 shows a trace before (above) and after the decompression of idle-times is performed. The markers between the traces indicate the compression.

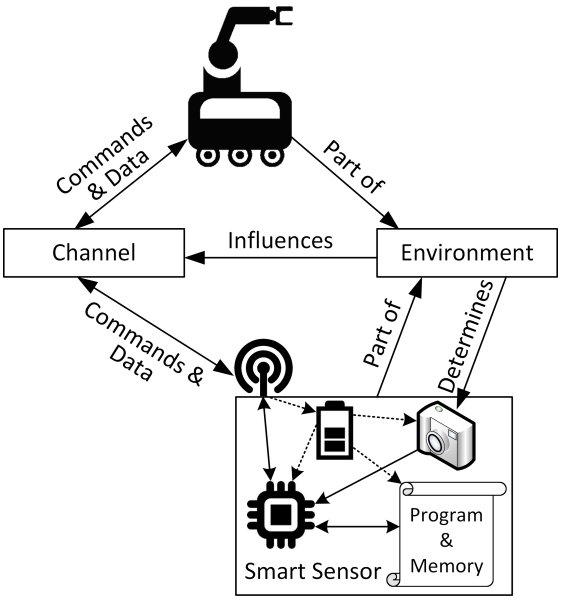

Fig. 5. Concept of the Evaluation Design.

The second challenge is the estimation of values that change during the skipping of time, such as the remaining energy in the battery. To estimate the values at the end of the time jump, detailed information about the process is required. As the time between two activations can be arbitrary, the error is unbounded. To mitigate that, a maximum time skip is defined. When linearizing the behavior of these transient values in the last instant, we can estimate the new value after the skip is completed. Depending on the maximum skip size, the final estimation can be very accurate.

4 Evaluation

The evaluation of the developed system is performed using the simulation of a smart sensor that charges the internal battery and communicates its information using NFC technology. The Gazebo simulator provides the context of the simulation. That is the environment in which the sensor is placed. During the startup of Gazebo, the developed sensor plugin is loaded and starts the SystemC simulation. A robot is placed outside the communication range of the sensor. The evaluation plan is to move the robot such that the sensor can be charged and communication can occur. When this is done, the robot requests data from the sensor.

For this evaluation we modeled the communication channel as a separate world plugin that can calculate the noises and signal attenuation due to the environment. This also allows us to change the channel parameters by swapping the

plugin. This also allows the reuse of the system for other communication technologies.

The communication between the two simulation environments is performed with strings encased in XML-tags. The received information is then stored in a data structure that groups the data blocks by the XML tag and stores the blocks in the order they arrive.

Different stimuli for the SystemC simulation can be combined, processed and sent by the developed plugin. Figure 5 shows the concept for this evaluation. The robot wants to send commands and data to the smart sensor via some channel (in this case NFC). To do so, it approaches the sensor, thereby changing the environment. This change influences the channel parameters. When the antennas are in close proximity to each other, data and energy can be transferred. The channel can, based on the parameters, change the data that is sent (e.g.: introducing bit errors). This message is then transmitted to the developed plugin. Furthermore, the plugin receives status information from the simulation, and parameters from the environment that the sensor can measure. This information is relayed to the SystemC simulation. The modified testbench processes the data to be used in the simulation. The simulation returns the results to the plugin. This plugin can manipulate the appearance of the sensor in the world (e.g.: switching on an LED), and transmitting the return messages to the channel. The channel again modifies the message according to the parameters and forwards it to the robot.

As the global simulation is done with a robotic simulator, a new test case can be implemented by changing the start position of the robot or introducing some randomness in the movement of the arm with the antenna attached to it. The rest of the simulation does not need to be altered in order to get new results. This simulation approach furthermore allows the testing of the interaction between the newly developed system and an existing (robotic) system. The evaluation of the correctness of the new system can also be done directly in the simulation as the robot expects certain answers.

As mentioned before, Figure 4 shows the compression of the idle time of the sensor showing the results of requirement R.9. Here the "advanceTime" trace is used to restore the time synchronicity of the two simulations after the simulation is finished. Whenever this trace peaks the simulation was stopped prematurely. The height of the peak indicates the time that is skipped. In this trace the first 7.1 seconds are condensed to about 0.6 ms. This means a reduction of memory and time usage of approximately 12000:1. Also the simulation can be stopped prematurely if the needed operations are finished before the time step is passed. This can be seen with the third drop of the "advanceTime" trace. This part is stored in 16.5 ms but refers to 0.15 s. This is a reduction of approximately 10:1.

Figure 6 shows the results of one simulation. Here the robot approaches the sensor until second 17. The antenna is activated from second 6.1 to 9. Here two messages are received (recData) but no return is generated, indicating that the channel has introduced errors in the sent message. This claim is substantiated by the fact that the received energy (EnergyIntake) is low and fluctuates. The fluctuation comes from the movement of the arm during the approach to the

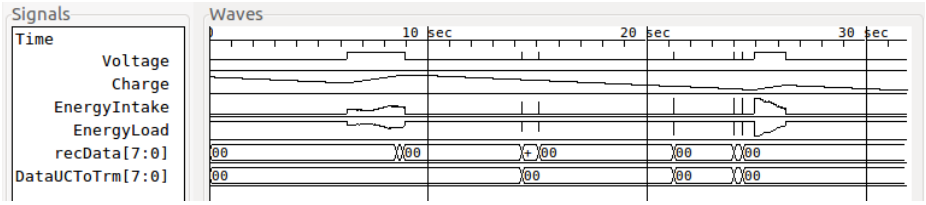

Fig. 6. Results of a completed simulation.

sensor.
The message at second 14.2 is being answered (DataUCToTrm). This means that the antennas are in a range where the communication can be successful. But the message at second 15.05 is not answered.
At second 21 the received energy is larger than before, indicating that the robot is close enough to communicate efficiently. In second 24, two messages are exchanged.
After second 25 the robot moves away. To show this the antenna is activated. During the retreat the received energy decreases.

The energy usage (EnergyLoad) of the sensor shows the combined energy budged of the sensor. As the energy gathered by the NFC antenna is much larger than the energy required for the calculations, it mirrors the energy intake. This also shows that the charging of smart sensors using NFC can be effective. The usage of the sensor itself can be seen in the trace of the remaining charge inside the capacitor (Charge). During the phases where the antenna is active, the charge rises, indicating that the capacitor is being loaded. In the mean time, the charge is slowly depleted as the sensor performs its operations with the energy stored in the capacitor.

Furthermore, the voltage available to the sensor (Voltage) is shown in this figure. Because the module used to charge the capacitor supplies the system with the maximum voltage allowed for this capacitor, the voltage rises rapidly to this value. In times where no energy is harvested the voltage is calculated using the charge of the capacitor, the energy currently used, and the serial resistance of the capacitor.

5 Future Work

A possible expansion of this system is another world plugin comparing the measurements of the sensor to the ground truth evaluated by the environment. This can be done by combining the information of the environment, the measured data from SystemC, as well as some of the messages received by the robot.
One drawback from forking the SystemC simulation from the Gazebo simulator is that the two processes are running on the same computer. If a simulation is created that encompasses multiple entities that are simulated using SystemC the simulations may need to share the same processor core, further slowing the

simulation. A solution to this would be to spread the SystemC simulation over a network and performing the communication using network sockets.

6 Conclusions

We presented an approach to connect SystemC simulations to the Gazebo simulator in order to automatically generate stimuli. This paper shows the difficulties that arise when connecting simulators that are designed to operate using different time steps. We showed a mechanism that can connect the simulations, proposed a mechanism that allows the interaction of the simulations, and formed requirements that need to be implemented on both sides to overcome the hurdles that we were presented by the simulations.
There are some core requirements that need to be changed we want to emphasize again. These include:

- *Synchronization* between the simulators is of utmost importance. SystemC operates usually more detailed and therefore needs longer to simulate one step. Gazebo must be halted while SystemC is running, otherwise the communication between the simulations can have unbounded delay.
- *Reduction of memory and time consumption* is important on all computers. Using our time reduction mechanisms, the realtime-factor of gazebo was optimized by a factor of 10^3 and the memory footprint reduced by up to 10^2.

This approach was first described by Pieber et al. [15]. In this publication, we extend the detail of the developed Gazebo plugin. Furthermore, we redefined some of the requirements needed for the SystemC adaptions to emphasize their purpose. Additionally, the evaluation describes how the systems interact and gives a detailed example of a complete simulation and what the created traces can look like.

Acknowledgements

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 May 2019. More information https://iktderzukunft.at/en/

References

1. Accelera: SystemC. http://accellera.org/downloads/standards/systemc (2000), last accessed on Jan 17, 2017

2. Bombana, M., Bruschi, F.: SystemC-VHDL co-simulation and synthesis in the HW domain. In: 2003 Design, Automation and Test in Europe Conference and Exhibition. IEEE Comput. Soc (2003)
3. Bouchhima, F., Briere, M., Nicolescu, G., Abid, M., Aboulhamid, E.: A SystemC/simulink co-simulation framework for continuous/discrete-events simulation. In: 2006 IEEE International Behavioral Modeling and Simulation Workshop. Institute of Electrical and Electronics Engineers (IEEE) (sep 2006)
4. Huang, K., Bacivarov, I., Hugelshofer, F., Thiele, L.: Scalably distributed SystemC simulation for embedded applications. In: 2008 International Symposium on Industrial Embedded Systems. Institute of Electrical and Electronics Engineers (IEEE) (jun 2008)
5. Kirchner, T., Bannow, N., Grimm, C.: Analogue Mixed Signal Simulation Using Spice and SystemC. In: Proceedings of the Conference on Design, Automation and Test in Europe. pp. 284–287. DATE '09, European Design and Automation Association, 3001 Leuven, Belgium, Belgium (2009)
6. Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566). Institute of Electrical and Electronics Engineers (IEEE) (2004)
7. Lee, W.S., Son, W.I., Oh, K.S., Yu, J.W.: Contactless energy transfer systems using antiparallel resonant loops. IEEE Transactions on Industrial Electronics 60(1), 350–359 (jan 2013)
8. Mathworks: Get Started with Gazebo and a Simulated TurtleBot. https://de.mathworks.com/help/robotics/examples/get-started-with-gazebo-and-a-simulated-turtlebot.html (2016), last accessed on Jan 03, 2017
9. Metta, G., Fitzpatrick, P., Natale, L.: Yarp: Yet another robot platform. International Journal of Advanced Robotic Systems 3(1), 8 (2006), https://doi.org/10.5772/5761
10. Meyer, J., Sendobry, A., Kohlbrecher, S., Klingauf, U., von Stryk, O.: Comprehensive Simulation of Quadrotor UAVs Using ROS and Gazebo. In: Noda, I., Ando, N., Brugali, D., Kuffner, J.J. (eds.) Simulation, Modeling, and Programming for Autonomous Robots. pp. 400–411. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
11. Mueller-Gritschneder, D., Lu, K., Wallander, E., Greim, M., Schlichtmann, U.: A virtual prototyping platform for real-time systems with a case study for a two-wheeled robot. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013. EDAA (2013)
12. Nurseitov, N., Paulson, M., Reynolds, R., Izurieta, C.: Comparison of json and xml data interchange formats: A case study. Caine 2009, 157–162 (2009)
13. Open Source Robotics Foundation: Gazebo simulator. http://www.gazebosim.org (2004), last accessed on Jan 03, 2017
14. Panda, P.R.: SystemC - A modelling platform supporting multiple design abstractions. In: Proceedings of the 14th international symposium on Systems synthesis - ISSS. Association for Computing Machinery (ACM) (2001)
15. Pieber, T.W., Ulz, T., Steger, C.: 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)
16. Possadas, H., Adamez, J.A., Villar, E., Blasco, F., Escuder, F.: RTOS modeling in SystemC for real-time embedded SW simulation: A POSIX model. Design Automation for Embedded Systems (2005)

17. Strommer, E., Jurvansuu, M., Tuikka, T., Ylisaukko-oja, A., Rapakko, H., Vesterinen, J.: NFC-enabled wireless charging. In: 2012 4th International Workshop on Near Field Communication. Institute of Electrical and Electronics Engineers (IEEE) (mar 2012)
18. Sumaray, A., Makki, S.K.: A comparison of data serialization formats for optimal efficiency on a mobile platform. In: Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication. pp. 48:1–48:6. ICUIMC '12, ACM, New York, NY, USA (2012)
19. Willow Garage and Stanford Artificial Intelligence Laboratory: Robot Operating System. http://www.ros.org/ (2007), last accessed on Feb 15, 2018
20. Wireless Power Consortium, et al.: System description wireless power transfer. Volume I: Low Power, Part 1 (2010)
21. Zamora, I., Lopez, N.G., Vilches, V.M., Cordero, A.H.: Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo. arXiv preprint arXiv:1608.05742 (2016)