

Integration of Heterogeneous Components for Co-Simulation

Jawher Jerray^a, Rabea Ameer-Boulifa^b and Ludovic Apvrille^c

LTCI, Télécom Paris, Institut Polytechnique de Paris, Sophia-Antipolis, France

Keywords: Heterogeneous Models, Simulation, Formal Verification, Integration, System Design.

Abstract: Because of their complexity, embedded systems are designed with sub-systems or components taken in charge by different development teams or entities and with different modeling frameworks and simulation tools, depending on the characteristics of each component. Unfortunately, this diversity of tools and semantics makes the integration of these heterogeneous components difficult. Thus, to evaluate their integration before their hardware or software is available, one solution would be to merge them into a common modeling framework. Yet, such a holistic environment supporting many computation and computation semantics seems hard to settle. Another solution we investigate in this paper is to generically link their respective simulation environments in order to keep the strength and semantics of each component environment. The paper presents a method to simulate heterogeneous components of embedded systems in real-time. These components can be described at any abstraction level. Our main contribution is a generic glue that can analyze in real-time the state of different simulation environments and accordingly enforce the correct communication semantics between components.

1 INTRODUCTION

Complex embedded systems are commonly designed using several modeling approaches and tools, because of the different nature of sub-systems, and because of the use of tiers to provide equipments. Integrating such heterogeneous is known as complex because of the diversity of models. Yet, ideally, this integration stage should be done as early as possible in the development process of these systems to verify, e.g., that the interfaces and main data exchanged are as expected, and can provide the expected overall functions.


Since forcing all partners of a product to use the same modeling languages or simulation techniques is a too hard constraint, integration of components designed within disparate formalisms necessitates a methodology for interconnecting these varying formalisms. A first way to do would be to glue the different meta-models of the components in order to build a unique (meta-)model from which integration verifications can be performed. This has already been shown in the scope of components for which their models of computation are quite similar (Zhao et al., 2020). Yet, when they are too different, a second approach


can be used: connecting these components at simulation level. This paper aims to address this challenge, concentrating specifically on data exchanges between components. The objective is to guarantee the enforcement of the correct communication semantics.


In this paper, we define a method and techniques to allow to integrate a set of models designed with different frameworks and simulated using their own simulator, with no modification on their simulation engine. To maintain a real-time co-simulation of these different simulators where they interoperate in realistic scenarios, we propose a generic "simulation glue" based on a distributed event streaming platform to join heterogeneous simulators together. . After having defined this glue, the paper illustrates in a more concrete way how SystemC and TTool components can be co-simulated, using Apache Kafka as the distributed event streaming platform.

2 RELATED WORK

In the area of heterogeneous distributed systems analysis, e.g., (Basu et al., 2006; Liboni and Deantoni, 2020), most of the works are based on co-simulation, but not many of them can support both simulation and formal validation in the same framework. For coupling two or more simulation tools in a co-simulation envi-

^a  <https://orcid.org/0000-0001-6170-7489>

^b  <https://orcid.org/0000-0002-2471-8012>

^c  <https://orcid.org/0000-0002-1167-4639>

ronment, most of the approaches (e.g., (Neema et al., 2014; Tavella et al., 2016; Mugombozi et al., 2019)) rely on the Functional Mockup Interface (FMI) Standard (Blochwitz et al., 2011) to bundle, in a single black-box, the internal computations and the interface descriptions of the simulation units. One of the most difficult challenges these approaches face is dealing with the gap between the different semantics (Tripakakis, 2015) of simulation units such as the various semantics of the coordination, which can refer to continuous time or discrete events. In (Liboni and Deantoni, 2020), the authors have defined a language for describing model coordination interfaces. The interface is dedicated to share the elements necessary to coordinate the execution and communication among the simulation units. There also exist dedicated platforms for the modelling and the simulation of heterogeneous component-based systems, examples include (Balarin et al., 2003; Eker et al., 2003; Basu et al., 2006). These platforms support several modeling languages with a variety of component semantics. However, they offer a general and unified framework for the design and simulation, even for hybrid systems. Among existing co-simulation solutions for integrating complex systems, the model transformation from a high-level language to SystemC is proposed in the scope of UML or SysML (Raslan and Sameh, 2007; ?; ?). In (Atitallah et al., 2008), the authors propose a tool to transform a high level MARTE description into an executable platform via a chain of model transformations. These solutions are relatively tedious since the transformation of high-level languages into traditional simulation or verification must preserve the semantics, which is commonly tedious for supporting correctly communication and computation semantics. Moreover, all modeling environments must propose a model transformation to the same target language (e.g., SystemC) and the simulation specifications obtained from high-level languages must still be integrated together.

3 SIMULATING HETEROGENEOUS MODELS

This section presents our contribution: a new framework for the integration of heterogeneous components using a simulation integration platform. This integration platform is agnostic to the simulation technology, as long as it supports the communication semantics described in this section.

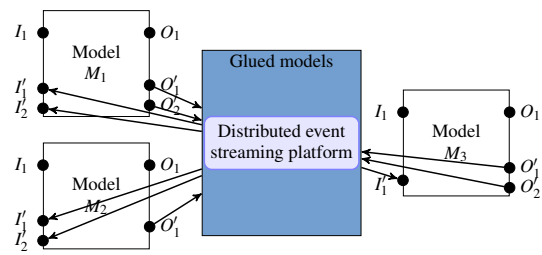


Figure 1: Communication between models and the distributed event streaming platform.

3.1 Component Model

A model defines components interconnected with ports. A model supports 4 types of ports: internal input ports I_i , external input ports I'_j , internal output ports O_k and external output ports O'_l . A model M_k has m internal input ports, n external input ports, p internal output ports and q external output ports as shown in Fig. 1. We assume that the objective is to co-simulate a set of components C . We assume that the simulation engine of each $c \in C$ can already handle communications internal to c . Thus, in this paper, we focus on external input ports I'_j and external output ports O'_l .

In this paper, we consider two modeling frameworks and their corresponding meta-models: TTool (TTo, 2023) and SystemC. The former can be used to capture SysML components that can be simulated with the internal simulator (Knorreck et al., 2009) of TTool to prove safety and performance properties. The latter uses components written in C++ and simulated with the SystemC simulation engine.

3.2 Distributed Event Streaming Platform

We assume that a facility, part of our contribution, can stream events to distributed senders/receivers so as to ensure data can be sent and received, in real-time, by the simulation engines of components.

Apache Kafka. Apache Spark, Apache Kafka, Apache Flink and Spring Cloud Data Flow are example of streaming platforms. We decided to rely on Apache Kafka to forward messages between simulation engines. Indeed, Kafka supports many computing platforms and can also handle distributed and event-based communications. Yet, Kafka cannot natively support the communication semantics usually found in modeling frameworks for embedded systems, for instance the exchange of values via FIFOs. Yet, Kafka supports the notion of *broker*: a broker

contains a set of topics, and each topic has a set of partitions that can be considered as an infinite FIFO buffer. To send messages to a partition, we rely on *producers*. A producer can send messages to different partitions in different topics. *Consumers* can receive messages from partitions. Partitions are configured at co-simulation setup.

3.3 Communication

In Fig. 1, we give an example of the simulation integration between 3 models via a distributed event streaming platform, where:

- Model M_1 has 2 external input ports I'_1 and I'_2 and 2 external output ports O'_1 and O'_2 .
- Model M_2 has 2 external input ports I'_1 and I'_2 and 1 external output port O'_1 .
- Model M_3 has 1 external input port I'_1 and 2 external output ports O'_1 and O'_2 .

The distributed event streaming platform must ensure communications, for example from external output O'_1 of M_1 to external input I'_1 of M_2 .

Fig. 2 depicts the general approach of our contribution. First, we assume that a user wants to co-simulate at least two models having possibly different meta-models. From those models, our contribution automatically updates these models to allow them to interact with our co-simulation framework. Then, using these Co-simulation models, we start the corresponding simulation of each model at the same time. We assume we can access to the simulation trace of each simulation when they are running: what is of interest for us is obviously to identify all the potential *read* or *write* transactions on external ports. Indeed, the co-simulator retrieves information that being sent or received on each port. When data is ready to be sent on an external output port, our co-simulator ensures data are forwarded to the corresponding external input port while enforcing the communication semantics, e.g., finite or infinite FIFO, exchange of values or exchange of a quantity of information.

Fig. 3 zooms on the co-simulation box shown in Fig. 2. The co-simulator is in charge of stream routing. For each model, the co-simulator manages the sending and receiving of data from the event streaming platform depending on the port type and the current status of ports.

3.4 Co-Simulation Models

The purpose of modifying the original models is to facilitate communication between the simulation of a model and the co-simulation. Co-simulation models

are automatically generated from the original model. The Co-simulation models are automatically generated from a given model.

Algorithm 1: Algorithm of the creation of the Co-simulation TTool model.

```

1 for each
  i_input_channel in list_input_channels do
2   disconnect_port_from_origin_side
     (i_input_channel); /* Disconnect the external
                        port from its origin. */
3   create_new_task(); /* Create new task for the
                        external input channel. */
4   connect_port_to_new_task(i_input_channel);
     /* Connect the external port to the new task. */
5   add_event(i_input_channel); /* Add an event
                                to the new task, this event will retrieve data from
                                the co-simulator using the avs command that allows
                                to add signals to a given event. */
6   associate_activity_diagram_to_task
     (i_input_channel); /* Create the activity
                        diagram of the new task that contains an infinite
                        loop with a read event to get the data sent by the
                        co-simulator followed by a write channel to insert
                        the data to the model. */
7   add_CPU_in_architecture(i_input_channel);
     /* Add a new CPU in the architecture and associate
                        the created task to it. */
8   link_CPU_to_bus(i_input_channel); /* Link
                                       the new CPU to the main Bus. */
9 end
10 for each
    i_output_channel in list_output_channels do
11   disconnect_port_from_destination_side
        (i_output_channel); /* Disconnect the
                            external port from its destination. */
12   create_new_task(); connect_port_to_new_task(i_output_channel);
        add_event(i_output_channel);
13   associate_activity_diagram_to_task
        (i_output_channel); /* Create the activity
                            diagram of the new task that contains an infinite
                            loop with a read event, followed by a read channel
                            to remove samples from the channel. */
14   add_CPU_in_architecture(i_output_channel);
        link_CPU_to_bus(i_output_channel);
15 end

```

Algorithm to Create a Co-Simulation TTool Model, as Shown in Fig. 2. A TTool model consists of two views: a functional view and a hardware view. In the functional view, SysML blocks are used to describe the functions and their communica-

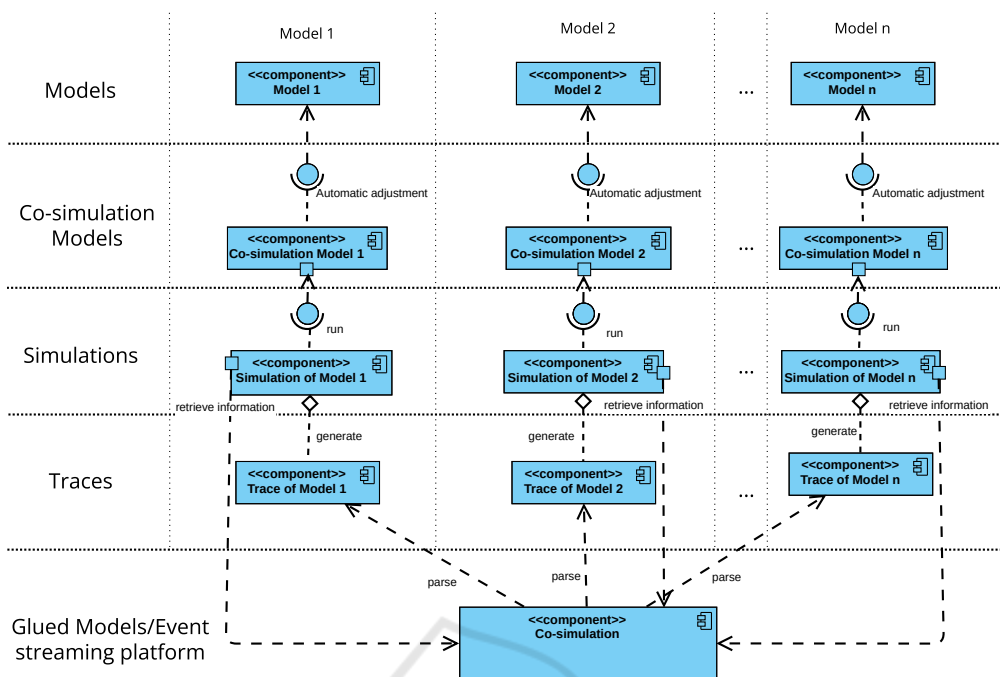


Figure 2: An overview of the proposed approach.

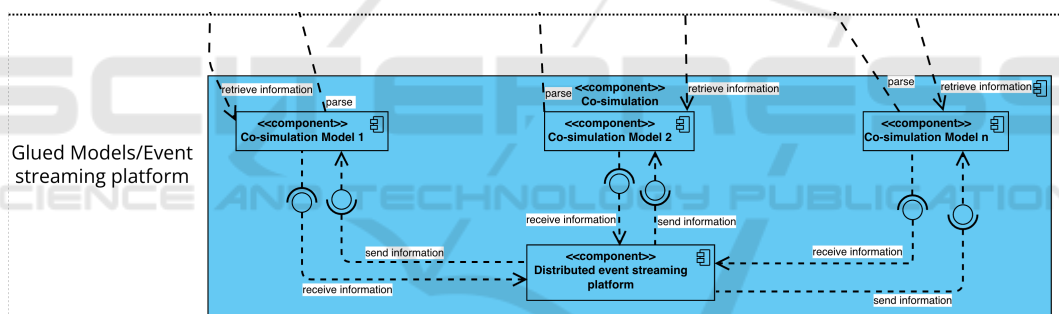


Figure 3: Zoom in the "Co-simulation" box.

tion. The behaviour of each function is given with an Activity Diagram. Functions can communicate together using two different facilities: event and data channels. Events can be used to exchange control values. Data channels are used to capture the exchange of a quantity of information. In the hardware view, functional blocks are allocated to blocks representing hardware components: processors, buses and memories. Functions are to be allocated to execution components (e.g., processors), and data communication are mapped to buses and memories. Events are not mapped since the traffic induced by control signals is usually considered as negligible. Events and data channels support different semantics, such as finite FIFO, infinite FIFO, etc.

Handling a communication (events, data) with a model external to TTool means that the hardware platform modeled in TTool must contain a input/output

hardware device from which all system components can be reached. This is how this is achieved in embedded systems: a communication interface must be used for input and output operations with the environment.

As a consequence, sending information to a component external to the TTool's model means that transactions on memories and on buses leading to the communication interface device must be taken into account when simulating the TTool model. Thus, when connecting TTool's simulator with another simulator, we decided to add one hardware I/O device per output or input channel, and the necessary connection to buses and memories of the hardware part of the TTool model. Moreover, we map to each hardware device one function f_{c_e} . The activity diagram of f_{c_e} features a main loop, which is infinite. In the case of an input device, the loop contains a wait event oper-

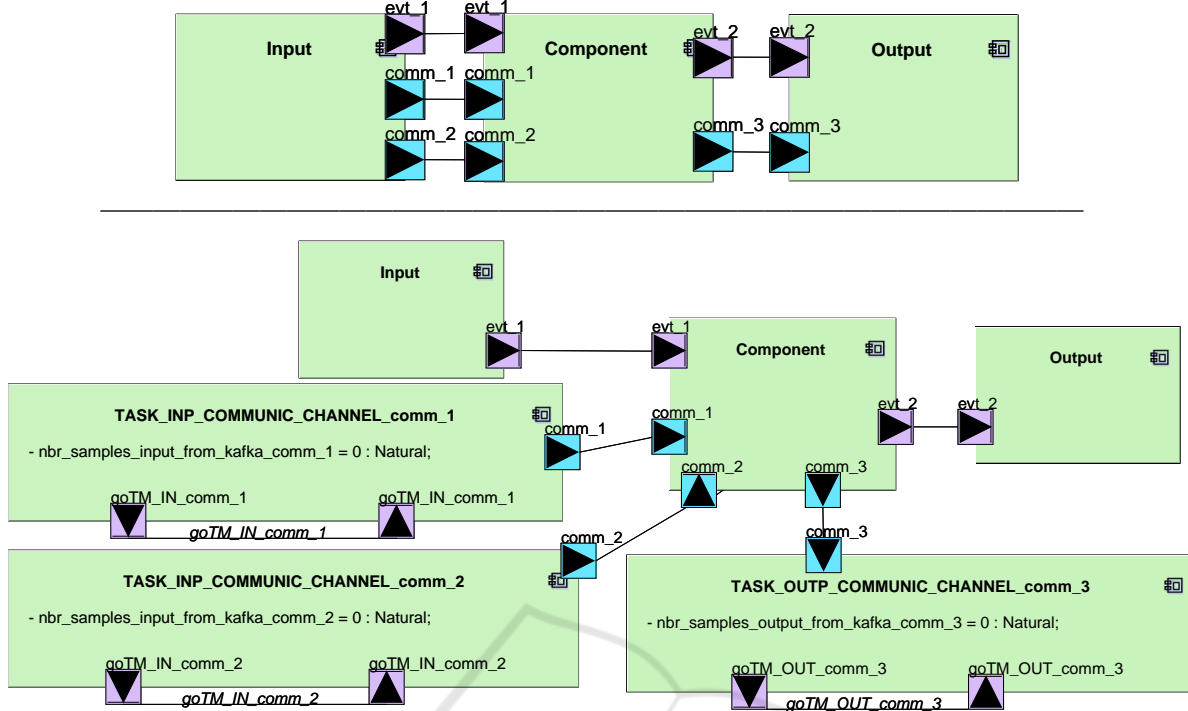


Figure 4: Top: The original Task diagram of the initial model. Bottom: The auto-generated task diagram of the Co-simulation model.

ator that makes it possible to receive the number of samples to be read (i.e., the quantity of information). The wait event is followed by a read channel operator. In the case of an output device, the infinite loop had a send event to send to the destination the number of samples to be read, and then a write operator. Algorithm 1 formalizes the different steps necessary to prepare a TTool model for external co-simulation: this includes creating the new functions f_{ce} and their behavior, creating the hardware I/O devices p_{ce} , connecting them to the corresponding bus, and connecting the new ports (ports of f_{ce}) to the corresponding sending ou receiving functions.

4 IMPLEMENTATION OF OUR APPROACH

An Example of Generating Automatically a Co-Simulation TTool Model. Let’s consider an example with 3 external input ports (“evt.1”, “comm.1” and “comm.2”) and 2 external output ports (“evt.2” and “comm.3”). The top part of the functional view in Fig. 4 and the top part of the architecture in Fig. 6 depict functional components and the allocation of components to the related architecture, respectively.

The application of Algorithm 1 to this model au-

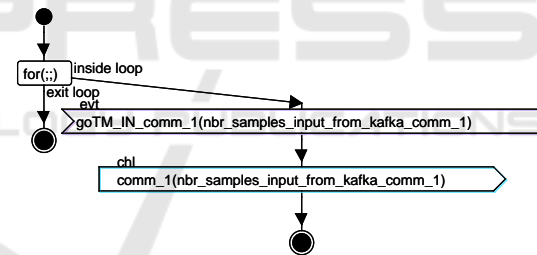


Figure 5: The auto-generated activity diagram for Channel $comm_1$.

tomatically transforms the component view to the one given at the bottom of Figure Fig. 4. Similarly, the activity diagram of the new task of the external channel “comm.1” is shown in Fig. 5, the activity diagrams of the tasks of the external channels “comm.2” and “comm.3” are similar to that of the channel “comm.1”, but its of “comm.3” has a channel read operator instead of the channel write operator. Also, bottom part of Figure Fig. 6 depicts the new hardware architecture.

Automatic Handling of Communication Semantics via Kafka. The general idea behind our contribution is to automatically create the necessary Kafka partitions to handle the communication semantics, e.g., to manage the number of events or data samples

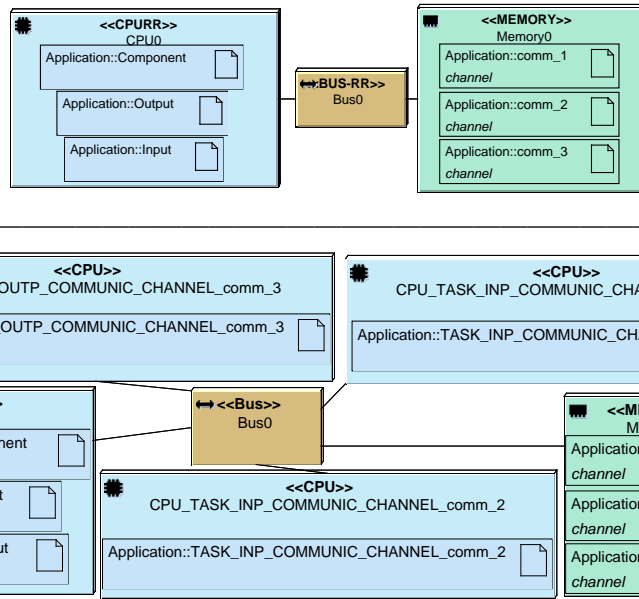


Figure 6: Top: The original architecture of the initial model linked to the top task diagram in Fig. 4. Bottom: The auto-generated architecture of the Co-simulation model linked to the bottom task diagram in Fig. 4.

in transit between two external ports. We associate to each component ports *consumers* and *producers*. A consumer intends to collect values from partitions, while a producer add information to partitions. Finally, our co-simulation framework is based on a set of partitions handled by Kafka and a set of producers and consumers.

We now review how the different communication semantics of data channels and events can be handled by the co-simulation framework ("Co-simulation" box in Fig. 2). For space reason, the paper focuses on one data channel communication semantics: *Blocking Read Blocking Write* (BRBW). An other communication semantics of type: *Block Read Non Blocking Write* with finite FIFO for events (BRNBW) can be found in section 4 of (Jerray et al., 2023).

Blocking Read, Blocking Write Channel. Let's consider a model built upon two components c_1 and c_2 exchanging data with an external communication. c_1 has an "external output" port *comm* of type "blocking write channel", c_2 has an "external input" port *comm* of type "blocking read channel". Because they are external, these two ports must exchange data via our approach to provide a blocking-read blocking-write channel communication. The approach is configured as follows.

Fig. 7 depicts the communication between c_1 and c_2 and presents the different producers and consumers that are used to enforce the correct communication semantics between them.

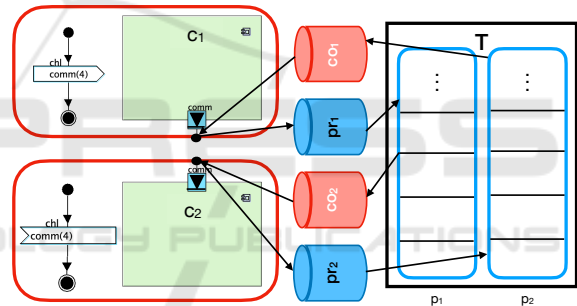


Figure 7: External communication between c_1 and c_2 via a BRBW channel.

First of all, a topic denoted T with two partitions p_1 and p_2 is created to apply the correct communication semantics between c_1 and c_2 . p_1 indicates the current number of samples that can be consumed by c_2 while p_2 contains the number of samples that has been already read by c_2 .

For c_1 , we create a producer pr_1 that puts in p_1 the number of samples that have been written. The producer pr_1 detects the written samples by analyzing in real-time the simulation trace of c_1 . c_1 relies on p_2 via consumer co_1 to know how many samples have been already read by c_2 to figure out how many samples can be transmitted to p_1 .

For c_2 , we create a consumer co_2 to get from p_1 the number of samples that has been written by pr_1 , and we use a producer pr_2 to send in p_2 the number of samples that have been read based on the simulation trace of c_2 . So, the consumer co_2 is blocked until a new element is added to p_1 or a read is performed by

c_2 .

Algorithm 2 features how the channels at output side (component c_1) are handled for a BRBW channel and Algorithm 3 shows how to handle channels at input side (component c_2) for a BRBW channel.

Algorithm 2: Algorithm of the output side (component c_1) for a BRBW channel.

```

1 search ←
  search_write_trans_in_trace(channel_name);
  /* Check if there is a write transaction of
  this output channel in the trace. */
2 if search is True then
3   nb_samples_written ←
     get_from_trace_number_of_samples_written
     (channel_name); /* Parse, from the
     trace, the number of samples of the
     external output channel that has been
     written. */
4   send_message( $pr_1$ ,  $T$ , 0,
     nb_samples_written); /* Send the number
     of samples in  $p_1$  of  $T$  created for this
     channel using its producer  $pr_1$ . */
5 end
6 nb_samples_read_by_c2 ←
  read_first_message_no_blocking( $co_1$ );
  /* Check if there is a new message in  $p_2$  since
  the last consumption and return the value of the
  first new message (element). */
7 if nb_samples_read_by_c2 != "" then
8   exec_command("avs "+
     name_of_event_created_for_channel_output
     + " 1 " + nb_samples_read_by_c2);
     /* Add the number of samples obtained from
      $p_2$  to the event of the new task that was
     created for the output external channel.
     Thanks to the avs command of the simulator
     that allows to add virtual signals for a
     given event. By adding the number of the
     samples read by  $c_2$  to the event a read
     transition will be succeeded to remove the
     obtained number of samples from the output
     channel buffer. */
9 end

```

Co-simulation of TTool SysML and SystemC models. We show how our approach can be used to co-simulate components designed in TTool (SysML) and others designed in SystemC. A ZigBee decoder serves as case study: we use the version described in (Enrici et al., 2017). ZigBee is a wireless communication scheme adapted to low-power devices.

Algorithm 3: Algorithm of the input side (component c_2) for a BRBW channel.

```

1 check_new_data ←
  check_new_data_in_topic(channel_name);
  /* Check if there is new data since the last
  consumption in  $p_1$  of the topic  $T$  that was
  created for this channel. */
2 if check_new_data is True then
3   nb_samples ←
     read_first_message_blocking( $co_2$ );
     /* Consume the first new message since the
     last consumption and return the number of
     samples found in  $p_1$  using the consumer  $co_2$ 
     created for this input channel. */
4   exec_command("avs "+
     name_of_event_created_for_channel_input
     + " 1 " + nb_samples); /* Add the number
     of the new samples obtained from  $p_1$  to the
     event of the new task that was created for
     the input external channel (for example the
     event goTM.OUT.comml in Fig. 5). By adding
     the number of the samples written by  $c_1$  to
     the event, a write transition will be
     succeeded to add the obtained number of
     samples to the input channel buffer. */
5 end
6 search ←
  search_read_trans_in_trace(channel_name);
  /* Check if there is a read transaction of this
  input channel in the trace. */
7 if search is True then
8   nb_samples_read ←
     get_from_trace_number_of_samples_read
     (channel_name); /* Parse, from the
     trace, the number of samples of the
     external input channel that has been read.
     */
9   send_message( $pr_2$ ,  $T$ , 1,
     nb_samples_read); /* Send the number of
     read samples in  $p_2$  of the topic  $T$  using the
     producer  $pr_2$ . */
10 end

```

In this example, we have divided the ZigBee decoder into 5 components (Source, symbol2ChipSeq, Chip2Octet, CW, and Sink) where Source, Chip2Octet and Sink are modeled and simulated by TTool while symbol2ChipSeq and CW are modeled and simulated using SystemC. All the external channels, in this example, are of type blocking read blocking write (FIFO of size 2) and the external events are of type blocking read, no blocking write with infinite FIFO. Also, we set the size of samples to 13.

After model improvement, our approach starts all

five simulation engines (one for each component), thus including all the necessary Kafka consumers and producers.

The description of case study and the results can be found in section 5 of (Jerray et al., 2023).

5 CONCLUSION

In this paper, we highlighted the need to integrate components together without common modeling languages nor heavy model transformations. Thus, we presented a method that allows to co-simulate, in real-time, embedded systems with heterogeneous components while respecting usual communication semantics between the components to be integrated. Our approach is based on simple model updates, on Kafka, and on the use of consumers and producers.

We have applied our method applies to mid-size systems such as Zigbee.

Having a distributed co-simulation has a cost in term of simulation time: we intend to lower the extra latency as much as possible: an option is to experiment with other brokers, even if Kafka has the advantage to be a recognized platform for distributed data exchange and is platform agnostic. We also intend to experiment with more modeling and simulation environments like AADL.

REFERENCES

(2023). TTool. <https://ttool.telecom-paris.fr>. [Online].

Atitallah, R. B., Marquet, P., Piel, É., Meftali, S., Niar, S., Etien, A., Dekeyser, J.-L., and Boulet, P. (2008). Gaspard2: from MARTE to SystemC Simulation. In *Workshop on Modeling and Analysis of Real-Time and Embedded Systems with the MARTE UML*.

Balarin, F., Watanabe, Y., Hsieh, H., Lavagno, L., Passerone, C., and Sangiovanni-Vincentelli, A. (2003). Metropolis: an integrated electronic system design environment. *Computer*, 36(4):45–52.

Basu, A., Bozga, M., and Sifakis, J. (2006). Modeling heterogeneous real-time components in BIP. In *Software Engineering and Formal Methods (SEFM)*, pages 3–12. IEEE Computer Society.

Blochwitz, T., Otter, M., Arnold, M., Bausch, C., Elmquist, H., Junghanns, A., Mauss, J., Monteiro, M., Neidhold, T., Neumerkel, D., Olsson, H., Peetz, J. V., Wolf, S., and Claub, C. (2011). The Functional Mockup Interface for Tool independent Exchange of Simulation Models. In *Proceedings of the 8th International Mod- elica Conference*, pages 105–114.

Eker, J., Janneck, J., Lee, E., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., and Xiong, Y. (2003).

Taming heterogeneity – the Ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144.

Enrici, A., Apvrille, L., and Pacalet, R. (2017). A model-driven engineering methodology to design parallel and distributed embedded systems. *ACM Trans. Des. Autom. Electron. Syst.*, 22(2):34:1–34:25.

Jerray, J., Rabea, A.-B., and Apvrille, L. (2023). Integration of heterogeneous components for co-simulation. working paper or preprint.

Knorreck, D., Apvrille, L., and Pacalet, R. (2009). *Fast Simulation Techniques for Design Space Exploration*, pages 308–327. Springer Berlin Heidelberg, Berlin, Heidelberg.

Liboni, G. and Deantoni, J. (2020). CoSim20: An Integrated Development Environment for Accurate and Efficient Distributed Co-Simulations. In *ICISE 2020 - 5th International Conference on Information Systems Engineering*, Manchester/Virtual, United Kingdom.

Mugombozi, C. F., Zgheib, R., Roudier, T., Kemmeugne, A., Rimorov, D., and Kamwa, I. (2019). Collaborative Simulation of Heterogeneous Components as a Means Toward a More Comprehensive Analysis of Smart Grids. In *2019 7th Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, pages 1–6.

Neema, H., Gohl, J., Lattmann, Z., Sztipanovits, J., Karsai, G., Neema, S., Bapty, T., Batteh, J., and Tummeseit, H. (2014). Model-Based Integration Platform for FMI Co-Simulation and Heterogeneous Simulations of Cyber-Physical Systems. In *Lund University*, pages 235–245.

Raslan, W. and Sameh, A. (2007). Mapping sysml to sys- temc. In *Forum on specification & Design Languages FDL*, pages 225–230. ECSI.

Tavella, J.-P., Caujolle, M., Tan, C., Plessis, G., Schumann, M., Vialle, S., Dad, C., Cuccuru, A., and Revol, S. (2016). Toward an Hybrid Co-simulation with the FMI-CS Standard. Research Report.

Tripakis, S. (2015). Bridging the semantic gap between heterogeneous modeling formalisms and FMI. In *Embedded Computer Systems: Architectures, Modeling, and Simulation SAMOS*, pages 60–69. IEEE.

Zhao, H., Apvrille, L., and Mallet, F. (2020). A model-based combination language for scheduling verification. In Hammoudi, S., Pires, L. F., and Selić, B., editors, *Model-Driven Engineering and Software Development*, pages 27–49, Cham. Springer International Publishing.