Co-Simulation of IP Network Models in the Cyber-Physical Systems Context, using a DEVS-based Platform

Julien Vaubourg, Vincent Chevrier, Laurent Ciarletta, Benjamin Camus

Co-Simulation of IP Network Models in the Cyber-Physical Systems Context, using a DEVS-based Platform

Julien Vaubourg
Inria, Université de Lorraine,
CNRS, LORIA, UMR 7503
Vandœuvre-lès-Nancy,
F-54506, France
julien.vaubourg@inria.fr

Vincent Chevrier
Université de Lorraine, CNRS,
Inria, LORIA, UMR 7503
Vandœuvre-lès-Nancy,
F-54506, France
vincent.chevrier@loria.fr

Laurent Ciarletta
Université de Lorraine, CNRS,
Inria, LORIA, UMR 7503
Vandœuvre-lès-Nancy,
F-54506, France
laurent.ciarletta@loria.fr

Benjamin Camus
Université de Lorraine, CNRS,
Inria, LORIA, UMR 7503
Vandœuvre-lès-Nancy,
F-54506, France
benjamin.camus@loria.fr

ABSTRACT

Cyber-Physical Systems (smart grids, cities, homes, etc) are composed of computing resources, actuators and sensors, connected through IP networks. These IP networks involve many technologies. In order to help designing and evaluating these systems, we are studying the modeling and simulation of IP networks in this context. Since there is no universal IP simulator proposing a model library corresponding to all the required technologies, we propose a solution to make different major IP simulators, generally not interoperable, interact with one another in a same co-simulation. Moreover, they should also interact with simulators corresponding to other fields of expertise involved in the simulation, mostly related to the physical or social aspects of these systems (e.g. power models, traffic models, weather models, etc). In this paper, we propose to address these issues as a multi-modeling problem, by integrating well-known IP network simulators into a DEVS-based co-simulation platform. We propose some concepts, helping to split a network topology into several models, to create input/output ports inside them, and to integrate them to a DEVS multi-model. We illustrate our solution thanks to a use case, including interconnected event-based models executed both by NS-3 and OMNeT++, equation-based models and the co-simulation platform MECSYCO.

1. INTRODUCTION

We are interested by the simulation of Cyber-Physical Systems (CPS), which have the specificity to mix models for communication networks and others areas of expertise, like power models, traffic models, weather models, etc.

Communication networks are almost entirely IP-based and include several different physical media and logical protocols. Many models for these technologies are provided by several off-the-shelf IP network simulators (*IP simulators*), but the implementation of all of the required models are not necessarily available for the same simulator, or available with the same accuracy. As a result, thanks to their own models library, different IP simulators can be complementary for modeling a network. We want to reuse existing models from different libraries, by modelling IP networks thanks to a multi-model (each model representing a part of the network), and executing it thanks to a co-simulation (involving different simulators, generally not interoperable). To achieve this goal, we have to propose solutions for some underlying issues, like splitting a network topologies into several models, defining input/output ports inside existing models, etc.

Moreover, the IP network models (*IP models*) have to interact with the other models corresponding to other fields of expertise involved in the CPS simulations. Therefore, the multi-model must also be able to integrate these heterogeneous models, dealing with different formalisms (e.g. event-based vs. equation-based) and the co-simulation must be able to integrate the corresponding simulators. This paper presents our solution, using a DEVS-based co-simulation platform.

The section 2 presents the common usages related to co-simulation with IP simulators and positions the goals and challenges of this paper. Section 3 introduces the concepts we defined for creating couplings between IP models (Spatial Couplings), then Section 4 introduces those defined for the couplings between IP models and others (Structural Couplings). Section 5 focuses on the multi-model creation & execution, thanks to DEVS and MECSYCO. Section 6 proposes some proofs of concept for validating our results. Finally, Section 7 illustrates with a use case corresponding to the simulation of a CPS, including equation-based models and an IP network modeled through several IP models libraries.

2. GOALS AND CHALLENGES RELATED TO CO-SIMULATIONS WITH IP SIMULATORS

We found two types of coupling with an IP model, in the literature: coupling with another IP model, and coupling with a model from a different field of expertise (*non-IP model*).

2.1 Couplings Between IP Models

Couplings between IP models are mainly used to distribute the execution of an IP model. Distributing an execution implies splitting an existing model designed for a specific simulator, into several models, for executing them over several instances of the same simulator. This is mainly done for performance reasons (speed of scalability). One of the main contribution [12] about this approach is the integration of the Message Passing Interface (MPI) standard for the IP simulator NS-3. MPI allows a modeler to create a complex model of the IP network and to define where the software environment is allowed to divide the execution into several parts, using separated simulator instances. Another work was done before with The Georgia Tech Simulator (GTNetS) [16], but it was an homemade IP simulator. This work was started after observing that attempting to backstitch scalability into an existing simulator is difficult, taking the case of PDNS [15] (distribution of NS-2 models) as an example.

With these works, the co-simulation is composed of several instances of the same simulator. In this paper, we are interesting in the coupling of different IP simulators together, for mixing different IP models libraries. Indeed, choosing an IP simulator for writing an IP model means to be restricted to the submodels (see Def. 1) available in its own models library. Usually, the IP simulators are not interoperable together. Consequently, the modeler cannot use submodels from different models libraries. Our goal is to split the network topology into several parts, and implement each part thanks to a different IP simulator, for being able to use several IP models libraries. We named this type of coupling a Spatial Coupling, as defined in Def. 2.

DEFINITION 1. *A **submodel** is a model provided by an IP simulator, thanks to its models library. Submodels can be connected together, enabling to model a complete network. They can be models for nodes, links, applications, etc.*

DEFINITION 2. *A **Spatial Coupling** corresponds to a coupling between two IP models, each one representing a part of a same network topology. A Spatial Coupling is linked to each model thanks to a input/output ports. It is in charge to ensure the IP packets exchanges between the two ports.*

The closest work is [17], with the coupling of GTNetS and PDNS thanks to an homemade co-simulation platform. It introduces two methods for doing couplings: cross-protocol and split-protocol stack methods. The first one corresponds to a topology split at the level of an equipment (e.g. a node or a link), whereas the second one corresponds to a topology split at a TCP/IP layer level (e.g. data link, transport, etc). In our case, we restrict the Spatial Couplings to what they named the cross-protocol stack method (considering the split-protocol stack method as a future research topic). However, as in this work, we focus on packet-level simulators that assume the packet is the atomic unit of data that is simulated.

In our case, we want to integrate already existing IP simulators, and they do not provide a generic solution for integrating them to a co-simulation, because the authors are also the ones creating the coupled simulators. Furthermore, their solution does not support couplings with non-IP models, required for CPS simulations and introduced in the next section.

2.2 Couplings Between IP Models and Non-IP Models

Couplings between IP models and models from other fields of expertise enable to represent a device inside a network and to use an external model for representing its application layer. With this type of coupling, for example, when an electric power meter is coupled with an IP network, the meter is modeled both in the power networks domain (e.g. with an ordinary equation-based model) and in the IP networks domain. The data received by the IP simulator from a coupled simulator has to be embedded in a message, to send from a source to a destination in the modeled network topology.

[8] offers a good synthesis of the different platforms combining power simulators with IP simulators. These papers choose to restrict their solutions to only one IP simulator. In our case, we want to find a solution not restricted to only one IP simulator, and couple IP simulators to any other models using different formalisms. We named this type of coupling a Structural Coupling, as defined in Def. 3.

DEFINITION 3. *A **Structural Coupling** corresponds to a coupling between an IP model, and a model α from a different field of expertise. With this type of coupling, α represents the application layer of a device, and is in charge to produce output data from input data. The device represented in α is represented in the IP model as a network device. The IP model has to be able to receive data from α and send it to another network device in the network. When data are received by the network device corresponding to α, it has to transmit them to the connected model.*

The next section deals with the time synchronization issues.

2.3 Time Synchronization

Lots of papers focus on the method to use for the simulators coordination. Furthermore, in the CPS context, the coordination method has to take into account the execution policies of the different simulators, using different formalisms.

This issue was solved in EPOCHS thanks to regular synchronization points, checking the state of the continuous components (power models) and checking if new events happened for the discret-event ones (IP models). Depending on the step size used for the continuous models, the simulation results will be more or less accurate. In 2007, [11] proposes to use the DEVS formalism, with the ADEVS platform [10] and a homemade synchronization algorithm, in order to improve the simulation results accuracy for the discret-events models. With this solution, each event is processed at the exact time it happens. In 2011, [9] proposes an intermediate solution, with the flexibility of EPOCHS but approaching the DEVS accuracy. With that solution, all simulators are in the same timeline and events are checked continuously, providing accurate

simulation results, with a reduced algorithm, but with very long execution times.

Many works focus on the use of HLA [4][6][7], and chose to conform their federates (simulators) to the HLA rules, using a RTI (simulation middleware) as a glue. HLA gives specifications describing how an RTI should working but not for how it should be implemented. The result is that available RTI implementations are often not compliant with the full HLA standard. The simulators having an HLA connector available are usually interoperable with a specific implementation of the standard. Finally, because there are many ways, with the shared objects, to do couplings between models with HLA, it is very difficult to couple models not designed to communicate together. More generally, HLA does not give answers for integrating models for different formalisms [18].

The details of our solution for the time synchronization are out of the scope of the paper. We solved this problem thanks to the co-simulation platform we chose to use and we contributed to. Some pointers are given in Section 5.

The next sections presents the concepts we defined, for creating Spatial and Structural Couplings.

3. SPATIAL COUPLINGS

3.1 Splitting a Network Topology

Splitting a network topology in order to represent different parts of it in different models requires to choose a place where to cut. We describe these places in this section, and propose a concept for representing input/output ports inside the models.

An example of basic IP model, composed of submodels, is proposed in Fig. 1. Two categories of submodels are presented here: nodes and links. Each of these submodels are composed of other submodels. In this paper, we chose to ignore the underlying submodels, except for the network devices composing a link, and the applications composing a node. We defined two places where we can split a network topology: End-Nodes (see Def. 4) and Transit-Nodes (see Def. 5). A node can be both an End-Node and a Transit-Node, e.g. a router acting as a proxy.

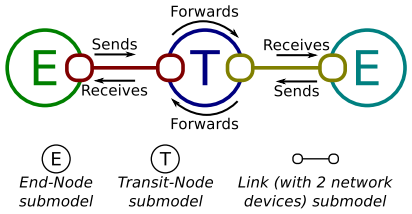

Figure 1. An IP model is composed of different submodels (in this example, 1 color = 1 submodel), with End-Nodes, Transit-Nodes and Links.

DEFINITION 4. *An **End-Node** is a node with applications. Incoming IP packets are intended for it, and outcoming packets are produced by it. It corresponds to the computer of an end user, or a server.*

DEFINITION 5. *A **Transit-Node** is a node without application. It is connected to several links, and its function is to forward IP packets from a link to another one. It has no packets directly intented for it. It corresponds to a router.*

As described in Fig. 2, splitting a simple network topology involves 3 places where to cut, leading to 7 combinations.

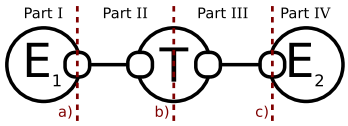

Figure 2. Splitting a network topology with a basic example and 7 possibilies (*a, ab, ac, abc, b, bc or c*).

Connecting the models together enables us to create a multi-model, representing the whole network topology. We consider that the data unit transferred between the models is the IP packet (corresponding to the layer 3 described in RFC1122). Thus, models have to exchange modeled IP packets (*IP packets*) together.

Connecting the models together requires to create bridges between them. When an IP packet reaches a splitting boundary inside a model, it has to cross the bridge for being transferred to the corresponding splitting boundary of the connected model. According to Def. 6, these boundaries are what we named Spatial Port Devices. According to Def. 2, bridges between two IP models are Spatial Couplings.

DEFINITION 6. *A **Spatial Port Device** corresponds to a modeled device (e.g. a network device), used for catching incoming IP packets and for injecting IP packets in the modeled network. They define the place in the network topology where the IP packets have to leave the current model, in order to be injected in another Spatial Port Device, defined in a connected model. These are also the place where the external IP packets sent by a connected model have to be injected in the network. During a simulation, a Spatial Port Device always exchange IP packets, in the two ways, with the same remote Spatial Port Device.*

3.2 Splitting on a Transit-Node

Splitting on a Transit-Node leads to a typical method for cutting the model.

We use the basic network topology presented in Fig. 2 as an example, considering that we want to split the topology into two parts. The cut occurs at the b line, on the Transit-Node. In this case, we have to represent the T node into two models X and Y. As described in Fig. 3, we turn the network device submodels of the two links on both sides of T, into Spatial Port Devices. The T node is represented in the two models, but with only one link attached to it.

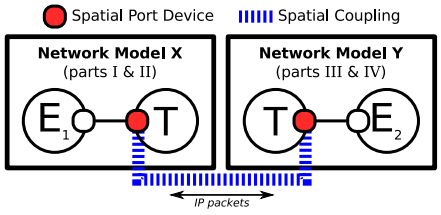

Figure 3. Splitting on a Transit-Node

In this case, E_1 knows that all packets for E_2 have to be send via T, thanks to its routing table. When E_1 sends an IP packet to E_2, this one is caught by the Spatial Port Device attached to T in X. Then, it is transferred to the Spatial Port Device attached to T in Y. Finally, it is injected in the network thanks to the routing table of T, for reaching E_2.

We supposed that we used static routing for this example. Once we split the network into several models, no model has a global knowledge of the whole topology. Consequently, it is impossible to let the IP simulators automatically fill the routing tables. This can be laborious when we want to model big and complex networks. Solutions for this problem are out of the scope of this paper, but [17] provides some strategies usable with our Spatial Couplings (e.g. using Ghost Nodes, static routes, well-known routing protocols like BGP, etc).

3.3 Splitting on a End-Node

Splitting on an End-Node leads to another typical method for cutting the model. With this type of splitting, we will see that some additional concepts are required.

We use again the basic network topology presented in Fig. 2 as an example, considering that we want to split the topology into two parts. The cut now occurs at the a line, on the End-Node E_1. We would like to have E_1 represented in a model X, and the other parts in a model Y.

In this case, the network topology represented in Y will be inconsistent, because of the link attached to T on a side, but to nothing on the other side. In fact, the part II of the topology cannot be represented without a node replacing E_1. For this purpose, we have to add a Fake Node to the model Y. We defined this concept in Def.7.

DEFINITION 7. *A **Fake Node** is a node added to a model for replacing a missing End-Node on a link, owing to a Spatial Coupling. It is just used for attaching the link and hosting a Spatial Port Device. It should not have any application and act as an empty shell. This node must have no influence on the simulation results.*

Based on our experience, creating a Spatial Port Device directly on an End-Node is difficult. Indeed, there is often a way to catch incoming IP packets but rarely to catch outcoming ones. Moreover, network device submodels are often dependent on a link submodel and cannot be separated from it. For this purpose, we need to add a link attached to E_1 for catching the IP packets sent by E_1. In order to be sure to not change the simulation results, this link has to be a Perfect Link, as we define it in Def.8. Because we need to have a node on each side of the link, we have also to attach a Fake Node on the Perfect Link.

DEFINITION 8. *A **Perfect Link** is a link attached to an End-Node on one side and to a Spatial Port Device on the other side, for catching the IP packets produced by the End-Node. It is useful when the IP simulator does not offer possibility to catch them directly on the node. This link must have no influence on the simulation results. This is why it should be represented thanks to a link submodel configured to have the maximum throughput allowed by the simulator and a null delay . Using a P2P link for that purpose avoids useless link layer messages, like ARP or NDP ones.*

The final multi-model is presented in Fig. 4. Thanks to the Perfect Link and the Fake Nodes, the simulation times associated to the IP packets are the same when E_1 sends them, and when the Fake Node in Y resends them.

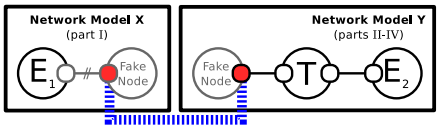

Figure 4. Splitting on a End-Node (Perfect Link represented with two parallel bars)

4. STRUCTURAL COUPLINGS

As explained in Def. 2, in the case of Structural Couplings, the data sent from the coupled model to the IP model corresponds to application layer data. Consequently, we have to define submodel corresponding to an application, and being able to act as a proxy between the network device and the coupled model. This is a new type of port that we call Structural Port Device and we define in Def. 9. An example of Structural Coupling is given in Fig. 5.

DEFINITION 9. *A **Structural Port Device** corresponds to a modeled application installed on an End-Node, used for receiving incoming application layer data from the network and transmitting them as input of a connected model. Structural Port Devices are also in charge to receive the output data of the connected model, then to send them to another network device. In the network, a Structural Port Device is connected to another network device, thanks to an UDP or TCP socket.*

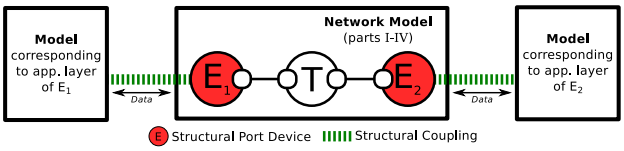

Figure 5. Structural Coupling.

The following section explains how we connect the models together, and we execute the multi-model.

5. CREATING AND EXECUTING A MULTI-MODEL

5.1 Co-Simulation With a Multi-Model

Thanks to the concepts defined above, we are now able to create connectors for IP models. We described different types of connectors (Port Devices), for different types of interaction (Spatial or Structural Couplings). These connectors enable us to provide input data to the models, and asking them for output data, during the simulation. The next step is to connect IP models and other ones together, for creating a multi-model. This multi-model will be executed in a co-simulation.

The following sections deal with the interconnection of the models, taking into account the heterogeneity issues, and the time synchronization.

5.2 DEVS Co-Simulation Platform

In order to connect the models together, and ensure the communications between the pairs of Spatial Port Devices and Structural Port Devices, we chose to use the DEVS Simulation Protocol [20].

Using a DEVS platform enables us to integrate different models with different formalisms, because DEVS is supposed to be able to integrate any other formalisms [18]. For executing our examples, we chose to use the DEVS-based MECSYCO co-simulation platform [2]. MECSYCO uses the Agents & Artifacts [14] paradigm for executing a multi-model, and the DEVS wrapper principle [13] to integrate models. Thanks to the Chandy-Misra-Bryant algorithm [3] and the multi-agent paradigm, the execution is fully decentralized.

In order to describe a multi-model, MECSYCO uses the following concepts. M-agents (Fig. 6a) are autonomous agents controlling the MECSYCO co-simulation, by handling the time advancement of a single individual simulator associated to a model. They also have the responsibility to retrieve the external events from their associated model and inject external events intended to it. An m-agent communicates with a model thanks to a model-artifact (Fig. 6b). This one corresponds to a DEVS wrapper attached to a model (Fig. 6c) and their associated simulator, and allows the attached m-agent to control the simulation. Finally, the coupling-artifact (Fig. 6d) ensures the exchange of events between the m-agents, putting them in a buffer, as a mailbox. Transformation operations may be used at the coupling-artifact level, for solving some multi-formalism or heterogeneity issues (e.g. changing time scales, adapting data units, etc).

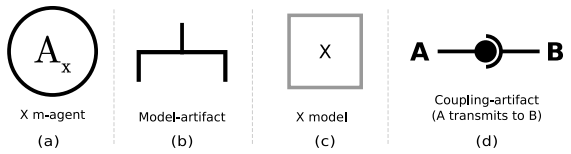

Figure 6. MECSYCO symbols.

Thanks to the DEVS concepts implemented in MECSYCO, we are able to create DEVS wrappers for our models and integrate them in the same co-simulation. MECSYCO helps use to solve heterogeneity problems, as demonstrated in [19]. Thanks to the possibility to add operations between the models, we can solve the time and data representation issues (e.g. we can convert the representation of the IP packets exchanged in a Spatial Coupling).

However, MECSYCO is a non-specialist platform and does not provide solutions for integrating IP models and simulators. For achieving that, we need to create DEVS wrappers. The next section deals with the prerequisites for a DEVS wrapper, and our concepts.

5.3 Interactions With the DEVS Wrapper

Our Port Devices can be seen as DEVS ports, with input and output events. Creating a DEVS wrapper requires that we are able to support four main functions:

- **Processing an external event:** An IP packet received as an external event has to be injected in the network by a Spatial Port Device. In the same way, an application layer data received as an external event has to be send on the network by a Structural Port Device.
- **Processing an internal event:** The next event recorded in the events stack has to be processed by the IP simulator.
- **Getting and output event:** The IP packet or the data received from the IP model, since the last internal event processing, has to be transmitted from the Port Device to the Wrapper.
- **Getting next internal event time:** The time associated to the next event recorded in the events stack, has to be transmitted from the IP simulator to the Wrapper.

Because some of these functions require to read and control the events stack, this implies to modify the events scheduler of the simulator.

The main goal is to "break the events-loop", in order to be able to provide a function enabling to execute the next internal event only when the DEVS wrapper asks it, for synchronization purposes. Based on our own experience, this step can be difficult, because all simulators are not written in a same manner. Simulators like NS-3 are convenient, because they use a single function with a loop inside (e.g. in the case of NS-3, it is `void Run()` defined in the headers of the `SimulImpl` class). Overriding it enables to add a mutex, which can be locked or unlocked from a function from another thread (corresponding to the DEVS wrapper). Other simulators like OMNeT++ do not provide a loop directly overridable, but core functions like the one returning the next internal event (e.g. in the case of OMNeT++, it is `cMessage* getNextEvent()` defined in the headers of the `cScheduler` class). Thanks to a mutex again and some efforts for avoiding deadlocks, the scheduler can also be instrumentalized.

The DEVS wrapper also requires functions for passing external events to the Port Devices, and retrieving the produced ones. For this purpose, when a Port Device is instantiated at the start of simulation, it has to be registered by the DEVS wrapper thanks to an index. During the simulation, the DEVS wrapper use its registry to exchange input and output external events with all available Port Devices (used for Structural Couplings as for Spatial ones).

The following section proposes a proof of concept, for validating the concepts presented above.

6. PROOF OF CONCEPT

This proof of concept (PoC) focuses on Spatial Couplings. Its goal is to demonstrate that we managed to exchange IP packets between models executed by two different IP simulators, without affecting the simulation results. The proposed scenario for our PoC is a modeled ping between two nodes connected together by a simple P2P link, then a LTE infrastructure (see Fig. 7).

A ping consists in exchanging packets, with a first node (*client*) sending a request (*echo message*) to a second one

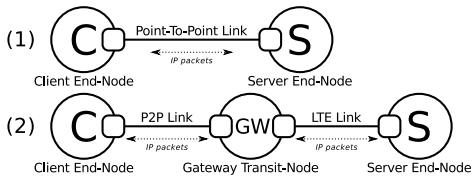

Figure 7. Network topologies corresponding to our PoC, modeling a ping between two nodes, via a P2P link and a LTE infrastructure.

(*server*). When the server receives an echo message, it sends a response (*echo reply*) to the client. Each node is enhanced with a modeled application to define its own role, with a ping client application (sending an echo message every second) for the client and a ping server (responding an echo reply for each received echo message) application for the server.

We want to split the first topology described in Fig. 7 on the server, corresponding to an End-Node. We want to split the second topology on the gateway, corresponding to a Transit-Node. Splittings are done exactly as described in Section 3, using a Perfect Link and Fake Nodes for the first topology.

We did the following ping simulations: 1) Model entirely written with NS-3, then with OMNeT++, 2) NS-3 model with the client and the P2P link, coupled to another NS-3 model with the Perfect Link (or the LTE infrastructure) and the server, 3) The same as the previous one, but with an OMNeT++ model on the server side and 4) Again the same configuration, but with NS-3 at the server side and OMNeT++ at the client side.

For validating our results, we did three groups of tests, corresponding to: 1) All the versions using only a P2P link, 2) All versions using an LTE network on the NS-3 side, and 3) All versions using and LTE network on the OMNeT++ side. The simulation time corresponding to the echo reply messages must be exactly the same for all versions inside a same group, experimentally showing that the coupling with MECSYCO (with Port Devices, Fake Nodes and Perfect Links) has no effect on the simulation results.

In order to compare the simulation results, we log the simulation time corresponding to the echo replies messages. We executed each version of our ping simulation an hundred of times, finding exactly the same simulation results. Then, we compared all versions inside a same group, and no difference was found, including with the versions without couplings (models entirely written in NS-3, then OMNeT++).

The next section proposes a use case corresponding to a complete simulation of a CPS, including a complex IP model with Structural and Spatial Couplings.

7. USE CASE

7.1 Scenario

Our scenario corresponds to the use case explained in [5]. Heatings and temperature sensors of two rooms are modeled thanks to an equation-based FMU [1] model (industrial standard way to model physical systems). Indoor temperatures are modeled taking account both the temperature transfered through the wall separating the two rooms, and the influencing outside temperature. The heat transfers are determined thanks to exchanges with an equation-based FMU modeling the wall, and another one modeling the outside temperature. Indoor temperatures can also be influenced by modeled heatings for each room, controlled with a variable representing a target temperature to reach.

As described in Fig. 8, a remote controller is added to this scenario. With this one, the user is supposed to be able to schedule target temperature changes (*heating controls*), depending on the time of the day. He is also supposed to be able to remotely access to a temperatures graph for each room (*monitoring*). For modeling this usage, a model of controller (Java application) is added, along with a model of IP network. The modeled IP network is described in Fig. 9: monitoring data are sent to the controller through the regular internet access of the house, and heating controls are sent by the controller through a dedicated LTE connection.

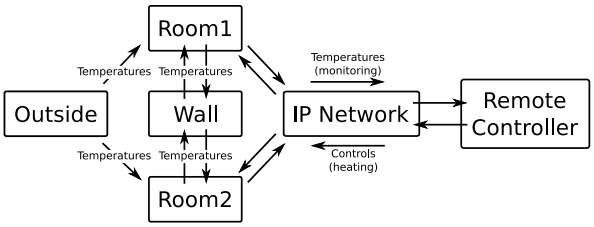

Figure 8. Intuitive graphic of the multi-model to compose.

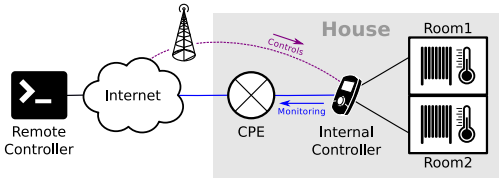

Figure 9. IP network topology to model.

In this experiment, we use Structural and Spatial Couplings with MECSYCO and the well-known IP simulators NS-3 and OMNeT++. We integrated these IP simulators to MECSYCO and we created libraries, including implementations of all of the concepts we described in this paper. Any submodel available for NS-3 or OMNeT++ can be used in a co-simulation, thanks to these MECSYCO libraries. Thanks to this example, we demonstrate that we are able to 1) interconnect models available in not interoperable IP models libraries, 2) test and compare similar models provided by different IP simulators, 3) easily test different types of networks with different topologies and 4) use an IP model for transporting data produced and used by models from another fields of expertise.

7.2 Modeling With a Single IP Network Model

As a first step, we model the whole IP network thanks to NS-3 (running on GNU/Linux, with C++ bindings). The NS-3 model is connected to the other models with Structural Couplings, as described in Fig. 10. The remote controller model and the FMUs are executed on another computers (running on Windows with Java bindings).

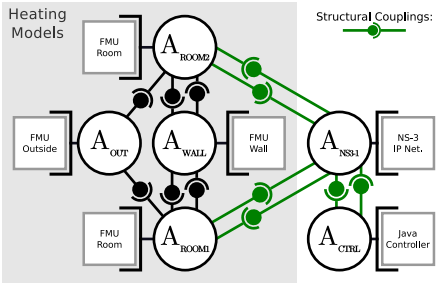

Figure 10. MECSYCO meta-model with only one IP model.

The IP Network modeled is described in Fig. 11. After some tests, the performance of the NS-3 LTE model raises questions. We would like to test another LTE model, from a different IP models library.

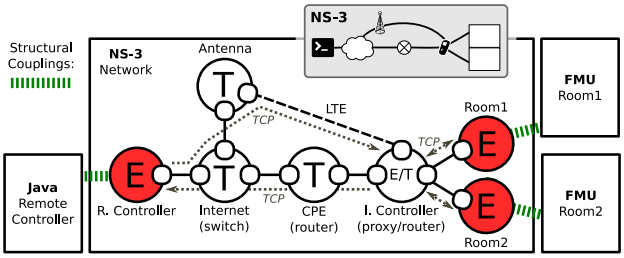

Figure 11. Modeled IP Network, with structural couplings on (E)nd-Nodes.

7.3 Adding an IP Model From a Different Library

In order to check if we could have better performance with another LTE model, we decide to try the model proposed in the OMNeT++ library. We add ports and spatial couplings, as described in Fig. 12 and we integrate an OMNeT++ (running on GNU/Linux, with C++ bindings) model in our multi-model, as shown in Fig. 13.

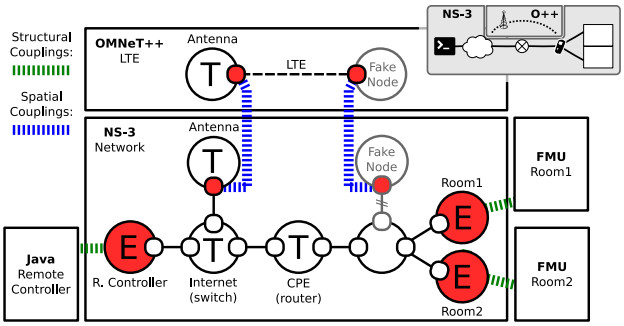

Figure 12. Modeled IP Network, with Structural and Spatial Couplings on (E)nd-Nodes and (T)ransit-Nodes.

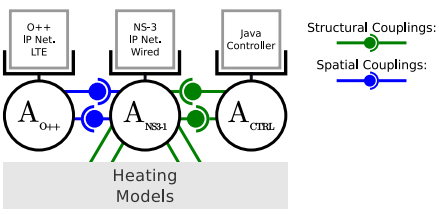

Figure 13. MECSYCO multi-model with two IP models.

In order to know if we improve the performance with OMNeT++ handling the LTE part, we compare the execution times of the simulations, thanks to a scalable version of our models with up to 20 modeled houses connected to the Internet. Results are available in Fig. 14. These results include a version with the LTE part modeled by a second instance of NS-3, for showing the current cost of the Spatial Couplings. With OMNeT++ handling the LTE part (with a model disabling periodic CQI feedback packets), we get better execution times.

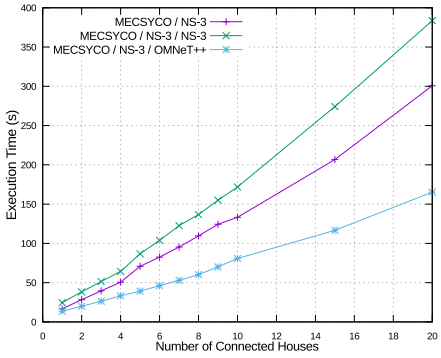

Figure 14. Performance comparison (using desktop computers).

As a last part of this experiment, we would like to test different technologies for the local area network (LAN), between the sensors and the internal controller.

7.4 Test of Different Models From the Same Library

In a first time, we split the IP model in order to isolate the LAN part in a distinct NS-3 model (Fig.15). Thanks to MECSYCO, we now can develop several versions of this part with different IP technologies (e.g. Wifi, PLC, etc), and interchange them quickly without any changes on the other parts of the multi-model. The resulted new multi-model is described in Fig. 16.

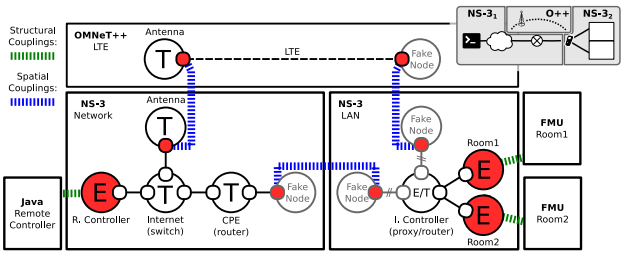

Figure 15. Modeled IP Network, with three IP models.

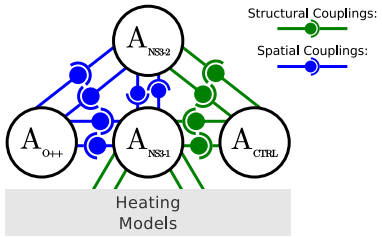

Figure 16. MECSYCO multi-model with three IP models (without model artifacts and models for readability).

Thanks to MECSYCO, we can choose to run the co-simulation with sensors connected to the internal controller by Wifi or PLC, just by launching the m-agent software corresponding the model implemented with Wifi or PLC, without any other modification elsewhere.

The final simulation results of your use case are shown in Fig. 17. These results are consistent with the ones provided by [5], considering that the remote controller was configured to send the target temperatures 293.15K (6 AM), 288.15K (8 AM), 293.15K (4 PM), 288.15K (10 PM) to the room 1 and 288.15K (10 AM), 293.15K (11 PM) to the room 2.

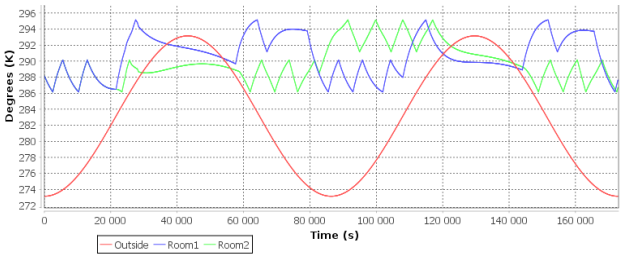

Figure 17. Simulation results of our use case (1 house and 2 simulated days).

8. CONCLUSION

The goal of this paper was to present our solution for modeling CPS, including physical models and IP models. We identified two types of coupling with IP models: Spatial and Structural Couplings. We then proposed some concepts for splitting IP models and connect them to models from other fields of expertise. Thanks to the DEVS formalism and the co-simulation platform MECSYCO, we demonstrate that we are able to connect the models together and to execute the multi-model.

We proposed some proof of concepts, experimentally validating that our couplings, using our concepts, have no influence on the simulation results. Then, we proposed a complete CPS use case, with equation-based models connected and an IP network topology modeled thanks to several IP models libraries, provided by different IP simulators. This experiment was an opportunity to demonstrate that using several IP models libraries is feasible thanks to our approach and can even improve the performance of the simulation.

As future works, we plan to work on the split-protocol stack method [17] and on a model driven solution for modeling complex IP networks using different IP models libraries.

ACKNOWLEDGMENTS

This work is partially funded by EDF R&D through the strategic project MS4SG.

REFERENCES

1. Blochwitz, T., Otter, M., Åkesson, J., et al. Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. *Proc. International Modelica Conference 2012*.
2. Camus, B., Bourjot, C., and Chevrier, V. Combining DEVS with multi-agent concepts to design and simulate multi-models of complex systems (WIP). *Proc. TMS/DEVS'15*.
3. Chandy, K., and Misra, J. Distributed simulation: A case study in design and verification of distributed programs. *Proc. TSE'79*.
4. Colby, S., and Beethan, D. Long range artillery simulation using component based development techniques and the high level architecture. *Proc. WSC'00*.
5. Gilpin, L., Ciarletta, L., Presse, Y., et al. Co-simulation solutions using aa4mm-fmi applied to smart space heating models. *Proc. SimuTools'14*.
6. Gottlieb, E., McDonald, M., Oppel, F., et al. The umbra simulation framework as applied to building hla federates. *Proc. WSC'02*.
7. Hibino, H., Yura, Y., Fukuda, Y., et al. Manufacturing modeling architectures: Manufacturing adapter of distributed simulation systems using hla. *Proc. WSC'02*.
8. Lin, H. Communication infrastructure for the smart grid: A co-simulation based study on techniques to improve the power transmission system functions with efficient data networks. *Thesis, 2012*.
9. Lin, H., Sambamoorthy, S., Shukla, S., et al. Power system and communication network co-simulation for smart grid applications. *Proc. ISGT'11*.
10. Nutaro, J. adevs: A discrete EVent system simulator.
11. Nutaro, J., Kuruganti, P., Miller, L., et al. Integrated hybrid-simulation of electric power and communications systems. *Proc. PES-GM'07*.
12. Pelkey, J., and Riley, G. Distributed simulation with MPI in ns-3. *Proc. SimuTools'11*.
13. Quesnel, G., Duboz, R., and Ramat, É. The virtual laboratory environment – an operational framework for multi-modelling, simulation and analysis of complex dynamical systems. *Simulation Modelling Practice and Theory. 2009*.
14. Ricci, A., Viroli, M., and Omicini, A. Give agents their artifacts: The A&A approach for engineering working environments in MAS. *Proc. IFAAMAS'07*.
15. Riley, G. PDNS - Parallel/Distributed NS, 2014.
16. Riley, G. F. The georgia tech network simulator. *Proc. MoMeTools'03*.
17. Riley, G. F., Ammar, M. H., Fujimoto, Richard, M., et al. A federated approach to distributed network simulation. *ACM Trans. Model. Comput. Simul. 2004*.
18. Vangheluwe, H. DEVS as a common denominator for multi-formalism hybrid systems modelling. *Proc. CACSD'00*.
19. Vaubourg, J., Presse, Y., Camus, B., et al. Multi-agent multi-model simulation of smart grids in the MS4SG project. *Proc. PAAMS'15*.
20. Zeigler, B. P., Kim, T. G., and Praehofer, H. *Theory of Modeling and Simulation*. Academic Press, Inc., 2000.