Co-Simulation of IP Network Models in the Smart Grids Context, using a DEVS-based Platform

Julien Vaubourg, Vincent Chevrier, Laurent Ciarletta

Co-Simulation of IP Network Models in the Smart Grids Context, using a DEVS-based Platform

Julien Vaubourg
Inria Grand Est
LORIA, Vandœuvre-lès-Nancy
F-54506, France
julien.vaubourg@inria.fr

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

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

ABSTRACT

We are interested in the modeling and simulation of the IP networks as the communication layer for smart grids. In this context, a network can involve many different technologies, and the available models corresponding to these technologies may be implemented in different simulation software. As a result, thanks to their own models libraries, different IP network simulators can be complementary. However, these simulators are not all interoperable with each other, and therefore can not be yet all integrated in a same co-simulation. Moreover, the network simulators have to interact with the other simulators corresponding to other areas of expertise involved in the smart grids simulations. Integrating this requires to consider the multi-formalism problems. Our approach is to integrate IP network simulators to the DEVS-based co-simulation platform MECSYCO (formely named AA4MM). Thanks to this approach, different network simulators can exchange simulated IP packets. In this paper, we illustrate how we integrated the NS-3 simulator into MECSYCO.

Categories and Subject Descriptors

C.2.0 [**Computer Communication Networks**]: General; I.6.m [**Simulation and modeling**]: Miscellaneous

General Terms

smart grids, IP, networks, multi-modeling, co-simulation, DEVS, MECSYCO, NS-3

1. INTRODUCTION

Smart grids are the next-generation electrical power systems, using communication networks for gathering information from the consumers, to improve the efficiency and the reliability of the production and the distribution of electricity. Smart grids are needed to prevent the global warming [5] or smooth consumption peaks. They are also necessary to meet other challenges like the impact of the democratization of the electric cars [17] on the power grid or the multiplication of actors operating on the grid. Smart grids include at least three main areas of expertise: power grids, communication networks and information systems. In this paper, we are interested in the simulation of the communication networks.

In the smart grids context, the communication networks are 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, but the implementation of all of the needed models are not necessarily available for the same simulation software (*simulator*). As a result, thanks to their own models library, different IP network simulators can be complementary for modeling a complex network. However, these simulators are not all interoperable with each other, and may use different formalisms.

Some available models may be specific to an area of expertise, may have been extensively studied, may have been extensively tested by the community, or may have a proven implementation. Rewriting the implementation of all necessary models for a chosen IP network simulator requires to have the skills in the corresponding area of expertise, implement the models without introducting errors and taking into account future evolutions. This may also be time-consuming or expensive. In order to avoid these problems and reuse existing models from different libraries, we would like to model a complex network with a multi-model interconnecting them together (each model representing a part of the network), and executed thanks to a co-simulation (involving different simulators). Moreover, the network models have to interact with the other models corresponding to other areas of expertise involved in the smart grids simulations. Therefore, the multi-model must also be able to integrate these models and the co-simulation must be able to integrate the corresponding simulators. This paper presents our solution.

Obviously, our solution must not have any impact on the simulation results, compared to the ones that should be obtained with execution of the same models, on a single simulator. It must be usable without requiring to change the simulator code or significantly modifying the models to integrate, to avoid introducing errors.

In order to integrate existing heterogeneous models and simulators to a same co-simulation, we need to choose a common formalism bringing them together. For several reasons

explained in the section 3, we chose DEVS (Discret EVent System) [24] as the common formalism. This article explains how we used the DEVS-based platform MECSYCO (Multi-agent Environment for Complex SYstems CO-simulation), formerly named AA4MM (Agents & Artifacts for Multi-Modeling [21]), for being able to model an IP network over several simulators.

The section 2 presents the common usages related to co-simulation with IP network simulators and positions the challenges of this paper. Section 3 rapidly introduces DEVS then the MECSYCO platform. Section 4 explains how we map IP network simulators with DEVS. Section 5 presents our solution for modeling a network over several simulators. Section 6 illustrates with a proof of concept showing an example of integration using NS-3 and MECSYCO. Section 7 proposes a discussion about this proof of concept.

2. COMMON USAGES RELATED TO CO-SIMULATION WITH IP NETWORK SIMULATORS

We identified two main usages in the literature, corresponding to two types of coupling of IP networks simulators. We named them structural and spatial couplings.

2.1 Structural Coupling

A structural coupling corresponds to the interconnection of two models, sharing a same state (set of variables representing a system at a given moment). The system dynamic is simultaneously represented through these two models. Each model can correspond to a specific area of expertise, describing a specific feature of the system.

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 (see Fig. 1). The data received by the IP network simulator from a coupled simulator is a message to send from a source to a destination in the modeled network topology.

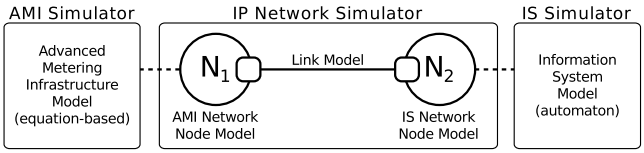

Figure 1: Example of structural coupling, with an electric power meter (AMI) connected to an information system (IS). They are represented both in two domains at once.

Structural couplings are often used in co-simulations involving heterogeneous formalisms. The main use-case proposed is the smart grids. [12] offers a good synthesis of the different platforms combining power simulators with IP network simulators. One of the most ancient work is EPOCHS [10] in 2006, linking two electromagnetic transient simulators with the IP network simulator NS-2. For the interoperability issues, it links multiple simulations into a distributed environment thanks to a homemade federated simulation system, in the spirit of the IEEE standard HLA [4].

The main problem studied is the time synchronization, with event-based IP network models and equation-based power network models. 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 network models). Depending on the step size used for the continuous models, the simulation results will be more or less accurate. In 2007, [15] proposes to use the DEVS formalism, with the ADEVS platform [14] 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. Another similar example with DEVS for a military application was proposed one year later by [11]. In 2011, [13] 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.

The second main usage is related to co-simulations with spatial couplings.

2.2 Spatial Coupling

A spatial coupling corresponds to the interconnection of two models, exchanging events as the corresponding modeled systems exchange data together. Therefore, an output (resp. an input) at the system level is represented by an output event (resp. an input event) at the model level.

With this type of coupling, for example, data exchanged between IP network models directly correspond to simulated IP packets (at least, a content with a pair of addresses) to transfer from a model to another, as illustrated in Fig. 2.

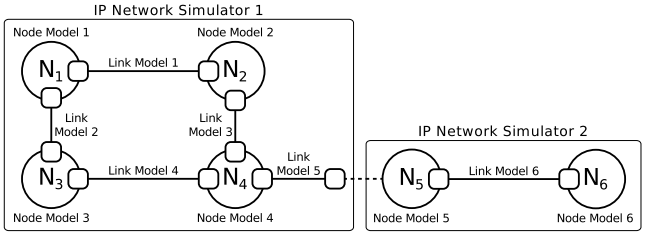

Figure 2: Example of spatial coupling, with IP network models exchanging IP packets.

Spatial couplings with IP network simulators are mainly used to distribute the execution of an IP network 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. One of the main contribution [16] about this approach is the integration of the Message Passing Interface (MPI) standard for the IP network 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. The

time synchronization is ensured by a conservative simulation algorithm. In this case, there is no multi-formalism issues because this solution is only restricted to multi-simulations with NS-3 instances connected together. Another strong limitation assumed by the authors is the need of point-to-point links within the IP network topology, which are the only places where the network can be cut. Another strategy was done before with The Georgia Tech Simulator (GTNetS) [20], a homemade IP network simulator. Although its goals are very close to the NS-3 ones, the project is oldest and was thought for the distributed simulations. This work was started after observing that attempting to backstitch scalability into an existing simulator is difficult, taking the case of PDNS [8] (distribution of NS-2 models with a federated simulation approach) in example. Several models of the internet protocol suite was redeveloped, and benchmarks with a half-million nodes topology were successfully performed.

With these works, the multi-simulation is composed of several instances of the same simulator, thus it is more distributions (top-down) than co-simulations (bottom-up).

The following section positions the challenges of this paper according to these two known usages.

2.3 Positioning

According to our requirements exposed in the introduction of this paper, the first part is to interconnect IP network models (executed by different IP network simulators) together in order to model the interconnection of IP network devices at the system level. As a result, the type of coupling required is a spatial one. However, the solutions studied above do not take into account the interoperability between different IP network simulators, with possibly different data representations.

The second part of our requirements is to be able to connect these IP network models to heterogeneous models from other areas of expertise, in the context of the smart grids simulation. As a result, the type of coupling required is a structural one. The time synchronization issues met in the cited papers, due to heterogeneous formalisms, are challenges.

In conclusion, the challenges related to our requirements are :

1. **Heterogeneous formalisms and representations integration [4]:** coupling IP network models with models using heterogeneous formalisms (e.g. event-based versus equation-based) and representations (e.g. time scaling).
2. **Simulation synchronization [6]:** controlling the time advancement of IP network simulators and managing the co-simulation, respecting the causality constraint and avoiding deadlocks.
3. **Software Interoperability [22]:** integration of IP network simulators to a co-simulation for enabling their models to exchange data together.

Our work is close to the integration of OMNeT++ to HLA by [7], that should be generic enough for be able to directly transmit IP packets through HLA to another simulator, although the interconnections was not studied for spatial couplings. In this paper, we present our solution for spatially coupling IP network models with potentially heterogeneous simulators, and we make choices taking into account the addition of structural couplings for the other simulators needed in the context of smart grids.

We choose to focus on existing IP network models, executed by event-based simulators that use an events stack for their internal processing. We consider that these models are composed of interconnected submodels, describing the equipments (devices also named nodes, links, stacks, network interfaces, etc), and used for simulating the packets transmissions. Finally, we consider that the IP network simulators can not be natively integrated in a co-simulation for exchanging together IP packets.

Integrating heterogeneous models and simulators in a same co-simulation requires to choose a common formalism for coupling them together. DEVS is such a formalism that can handle these issues and introduced in the next section.

3. CO-SIMULATION BASED ON DEVS FORMALISM

3.1 The DEVS Formalism

The DEVS formalism is the most general formalism for discrete event model [24].

A DEVS atomic model is a structure including at least:

- a set of states,
- a time advancement control,
- sets of input and output values with associated ports,
- a dynamic, evolving through internal events and generating external events, and
- an external transition function, changing the model state depending on external events and scheduling internal events.

A DEVS coupled model is composed of interconnected DEVS atomic models and has the same properties than a coupled model.

DEVS offers a simulation protocol for simulating a coupled event-based model and this protocol uses four kind of messages (see Fig. 3 from [24]):

- initialization messages `(i,t)` (`t` corresponding to the current simulated time), received by the simulator before the co-simulation starts up,
- internal transition messages `(*,t)`, to schedule internal events and changing model states,
- output messages `(y,t)`, to notify the other simulators an external event,

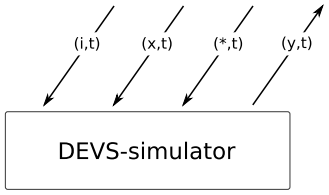

Figure 3: The DEVS simulation protocol.

- input messages `(x,t)`, to receive an external event from other simulators.

In our case, the IP network simulators have to interact with the other simulators corresponding to other areas of expertise involved in the smart grids simulations. Integrating these areas of expertise requires to consider the multi-formalism problems. According to [23] and [18], it's proven that all formalisms can be integrated in the DEVS formalism. Using a DEVS-based platform for integrating IP network simulators to a same co-simulation enables to prevent the multi-formalism problems.

As explained in [2], the DEVS formalism has some limitations about interoperability between simulators. Several DEVS-based platforms exist with a workable implementation, for building co-simulations. Among these ones, the MECSYCO platform solves these limitations, with a set of concepts introduced in the next section.

3.2 MECSYCO

MECSYCO offers Agents & Artifacts [19] concepts for handling interoperability between simulators, and is based on the DEVS formalism for integrating models with different formalisms [2].

With MECSYCO, the multi-model is seen as a DEVS coupled model and each model (or submodel) from the multi-model is controlled as a DEVS atomic model. Thanks to the DEVS wrapper principle [18], non-DEVS models can be integrated. Wrappers emulate the DEVS simulation protocol.

Thanks to the Chandy-Misra-Bryant algorithm and the multi-agent paradigm, the execution is fully decentralized and both the deadlock resilience and the causality constraint compliance are proven [3][1].

In order to describe a multi-model, MECSYCO uses the following concepts. M-agents (Fig. 4a) 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. 4b). This one corresponds to a DEVS wrapper attached to a model (Fig. 4c) and their associated simulator, and allows the attached m-agent to control the simulation. Finally, the coupling-artifact (Fig. 4d) 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 the time scale, adapting the data unit, etc).

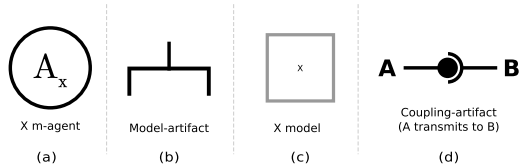

Figure 4: MECSYCO symbols.

The MECSYCO platform is implemented in Java and C++.

We consider that the IP network simulators on which we focus are non-DEVS compliant. As a result, we need to write DEVS wrappers for the integration within MECSYCO. The next section deals with the mapping of non-DEVS models to the DEVS formalism, with a focus on IP network models.

4. MAPPING OF IP NETWORK SIMULATORS WITH THE DEVS FORMALISM

4.1 Models Mapping with the DEVS Formalism

To integrate non DEVS-compliant IP network models with the DEVS-based MECSYCO platform, we need to map the DEVS concepts to their native ones.

According to the list of concepts for a DEVS atomic model provided in the section 3.1, we have to map:

- **A set of states:** we consider that the state of our models corresponds to the set of the state variables composing the model and its submodels (at least the properties of the modeled devices, links and packets).
- **Sets of input/output values:** In the case of spatial couplings, we need to exchange simulated IP packets between models. The IP packets collected in a model are intended to be injected in another model of the multi-model. Thus, we need to determine specific locations in the modeled IP network topology, for injecting external IP packets (input events) and collecting internal IP packets (output events). This is the port concept, and if the models don't have natively some kind of port we have to define and implement it (see section 5.1).
- **A dynamic:** with IP network models, the dynamic corresponds to the transmission of simulated IP packets, from a modeled equipment to another one in the model. Depending on the simulator used, the IP packet transmissions will correspond to one or several internal events (e.g. transmission from one device or link to another, from one IP layer to another, etc). A packet transmission between nodes in the IP network model can lead to an external packet production, then caught by a DEVS output port.
- **An external transition function:** when an external IP packet is injected within the model, somewhere in

the modeled IP network (via a port), the model state has to evolve in reaction. In our case, this corresponds to the scheduling of internal events, in order to simulate an IP packet sending, from the IP address of a modeled equipment to another one. The corresponding IP address and the packet content are found in the data associated to the external event received.

Once the non-DEVS model is enhanced by a DEVS wrapper for ensuring the DEVS compliance, the simulator executing the model has also to be DEVS-compliant. Therefore, we need to complete the features of the developed DEVS wrapper. The next section focuses on the mapping of non-DEVS IP network simulators, thanks to the DEVS simulation protocol.

4.2 Mapping of IP Network Simulators With the DEVS Formalism

As considering in the section 2.3, the IP network simulators on which we focus use an event stack for their internal processing.

According to the list of messages defined by the DEVS simulation protocol (cf. section 3.1), integrating a simulator to a co-simulation requires:

1. **Event processing interruptions and control:** the simulator has to be able to interrupt its event processing to receive input messages `(x,t)`, corresponding to the arrival of a new external event to process. This new event could be more recent than the next internal event ready to be processed, thus it is necessary to respect the causality constraint. For simulators not designed for co-simulations, this implies modifying the implementation of the event-loop.
2. **Event prediction:** the simulator also has to be able to return the time of the next event, in order to enable the m-agent to produce its internal transition messages `(*,t)`, through its associated model-artifact. Thus, at the implementation level, it needs to have an interface to retrieve the time of the next event ready to be handled by the event-loop.
3. **External event generation:** the simulator must be able to generate output messages `(y,t)` when one of its ports is concerned by the production of an external event. The content associated with this event must also be stored and delivered with the output message.
4. **Initialization:** before the co-simulation startup, a co-initialization can be necessary (e.g. for adapting the modeled IP network topology according to other models connected through the co-simulation). So, the software simulator has to be able to receive initialization messages `(i,t)` before starting its own simulation. At the implementation level, this can imply to add some requests before the event-loop starts.

The external event messages processing requires to have DEVS ports integrated at the model level and at the simulator level. Details about the port definition and integration in a non-DEVS model executed by a non DEVS-compliant IP network simulator, are given in the next section.

5. NETWORK MODELING OVER VARIOUS SIMULATORS

5.1 Splitting The Network

Splitting an IP network in order to model it over several simulators requires to choose a network node in the topology where the separation will occur, as shown in Fig. 5.

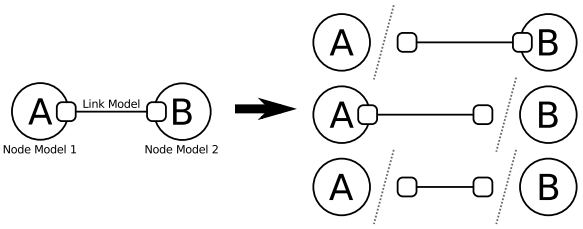

Figure 5: Splitting a model with three main submodels (two nodes and a link): three possibilities.

The IP network is cut into two parts, so we need to add a mechanism for linking them during the simulation. We introduced the DEVS port concept in the section 4.1, describing a port as being in charge of handling input and output events. Therefore, we need to create a pair of ports in the IP network, to link the two parts together. These ports will act as gateways for the simulated IP packets, and will ensure the transmission of the corresponding events between the models, thanks to the DEVS simulation protocol.

Considering that the models are composed of submodels modeling each part of an IP network (e.g. IP equipments with applications, network devices, packet queues, etc), a port can be implemented at many levels.

According to our requirements described in the introduction, three main ones should be taken in account for the port integration:

1. The port mechanism should not have any impact on the simulation results (the simulation results should be the same, with an IP network modeled through a single model, as modeled through a multi-model with models executed by different simulator instances).
2. The IP packets to transmit as external events must be caught at the lowest level, regarding the OSI model layers, in order to lose as little information as possible.
3. The port integration must have the minimal impact on the original model, in order to be able to easily reuse existing ones, avoiding introduction of errors.

The port integration at the implementation level is introduced in the next section, with the Port Node concept.

5.2 Port Node Concept

A port at the IP network level has to be a place where the simulated packets can be routed or switched. As a result,

an output port is a place where IP packets can leave the current network, and an input port is a place where external IP packets can be injected in the current network. When we couple the two models corresponding to the two parts of the network, we link output ports from the first model with input ports from the second model, and vice versa. In this way, the simulated IP packets are transmitted from a modeled network part to the other.

The solution we propose is to create Port Nodes in the modeled IP network, for representing the topology cuts and acting as bridges for the rest of the IP network. The Port Node is a basic network node and is like an empty shell, without application and without any internal event scheduling impacting the simulation results. The Port Node is able to inject simulated packets in the IP network and store the received ones to transmit it to the DEVS wrapper. For that, the network device model is the best place for catching IP packets at the lowest level, so the Port Node needs listening its network interface and be able to directly inject raw packets.

Port Nodes have to be linked to the IP network of the model with link models. This is the topic of the following section, introducing the Perfect Link concept.

5.3 Perfect Link Concept

For splitting the IP network by isolating a complete part of it, we can choose a node where the network is to be cut (corresponding to the two first possibilities illustrated in Fig. 5).

For example, we need to split into two models ($\mathcal{M}_1$ and $\mathcal{M}_2$) a modeled network, with a node $\mathcal{A}$ connected to a node $\mathcal{B}$ thanks to a link $\mathcal{L}$. The modeler can use the link $\mathcal{L}$ to couple a Port Node to the node $\mathcal{A}$ in $\mathcal{M}_1$. For the second Port Node connected to $\mathcal{B}$ in $\mathcal{M}_2$, the link does not exist in the system – as the Port Nodes – and must therefore be transparent. It should have infinite performances in order to have no influence on the simulation results. We named it a Perfect Link.

In fact, it should ideally have the maximum throughput and the minimum delay allowed by the simulator, with the minimum latency possible. Using a point-to-point link for the Perfect Link avoids useless link messages (ISO layer 2), like ARP or NDP ones.

When the modeler chose to isolate a link model, as in the last possibility illustrated in Fig. 5, he has to use a Perfect Link on each side. The process for connecting two separate networks together in order to model the system, with Port Nodes and a Perfect Link, is described in Fig. 6.

In the Fig. 6, the Port Node attached to "B" must have the same IP address as "C" and the Port Node attached to "C" must have the same IP address as "B". This constraint is necessary for maintaining consistent routing tables, in the models. The next section deals with this kind of route calculation issues implied by the IP network splitting.

5.4 Route Calculations

A common approach for routing with off-the-shelf simulators is to calculate the best path from any node to all other ones

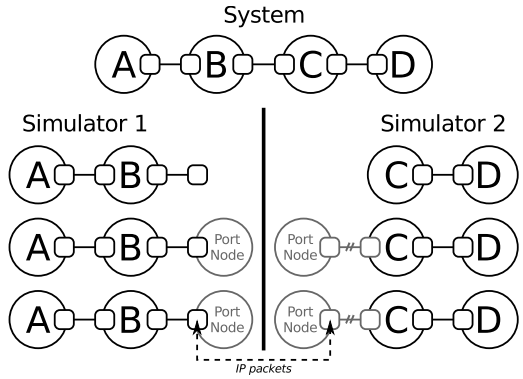

Figure 6: Network topology modeled over two simulators thanks to Port Nodes and a Perfect Link (with two parallel bars).

before the simulation starts, and fill all individual routing tables from all nodes modeled in the IP network. With a modeled IP network divided into several models, executed over several simulator instances, no simulator has a global topology knowledge. Thus, no route will exist in the routing tables for the nodes modeled in a remote model.

Therefore, splitting an IP network requires to find solutions for keeping the global routing table up to date.

[20] proposes two solutions for solving this issue:

1. The modeler statically associates the full list of the reachable IP addresses behind each coupling, for each Port Node. This full list could be directly translated into static routes and added to the Port Node routing table.

2. The entire topology is modeled in each model, with Ghost Nodes. These nodes are reduced state objects, containing only connectivity information used for routing decisions, and are linked with simple point-to-point links.

In a first time, we chose the second solution, considered more convenient.

The next section presents a proof of concept, with the simulation of an IP network over several simulator instances, and the integration of a non-DEVS compliant simulator.

6. PROOF OF CONCEPT

6.1 Goals

The goal of this proof of concept is to demonstrate that we managed to exchange IP packets between the models of two or more instances of a network simulator, without affecting the simulation results. We chose the network simulator NS-3 for this proof of concept. The implementation is generic enough to theoretically use NS-3 coupled with another IP network simulator.

The proposed use-case for our proof of concept is a modeled ping between two computers (nodes) connected together by a simple point-to-point link, as shown in Fig. 7.

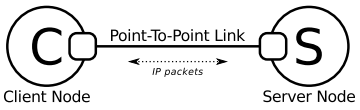

Figure 7: Network topology corresponding to our proof of concept, modeling a ping between two nodes.

A ping consists in exchanging packets, with a first node (*client*) sending a request (*echo message*) to the second node (*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 use NS-3 as the simulation software, because it is a popular simulator in the IP networks research field and is not compliant with DEVS. The client and server sides are modeled with the same simulator, in order to be able to compare the simulation results of the two-simulator instances version versus the one-simulator one. Nevertheless, our solution must be generic enough to use NS-3 coupled with another IP network simulator.

6.2 NS-3 Overview

NS-3 (Network Simulator 3) [9] is an IP network simulator existing since 2008 and co-developed by Inria Sophia Antipolis, the Washington University and the Georgia Institut of Technology and many other contributors.

Its predecessor NS-2 has been used extensively in research papers for many years, according to Fig. 8. NS-3 is a full rewriting of NS-2, without retrocompatibility, and this latter is no longer supported. The NS-3 working is very close to a Linux kernel: each layer of the TCP/IP model is modeled and a simulated IP packet has the same format that an IP packet in the real world.

NS-3 does not provide external ways to instrument it. The next sections detail the integration of NS-3 to the DEVS-based platform MECSYCO, then the proof a concept exploiting this integration.

6.3 Network Device Wrapper Concept and DEVS-compliant Coordinator with NS-3

To meet our challenges with NS-3, we developed a C++ library thanks to the C++ version of MECSYCO and the NS-3 framework.

This library is a MECSYCO extension for NS-3, including the model-artifact (DEVS wrapper), with the following features:

- **Port Node implementation:** a Port Node in NS-3 can be represented with a basic node model (without modeled applications installed on it). It must be connected to the rest of the network with a link model and registered as a Port Node thanks to the developed library. Registering a network device model as a Port Node creates a Network Device Wrapper, using the NS-3 features to directly catch the incoming simulated packets and inject external packets in the simulated IP network. Incoming packets in the Port Node are stored as output external events. Each Port Node is associated to an ID, enabling the model-artifact (DEVS wrapper) to know how to process the external events produced from and to that Port Node.

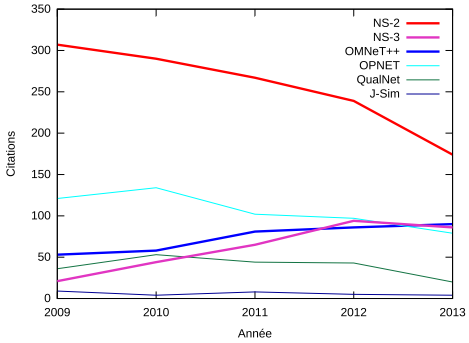

Figure 8: Number of papers proposed by the search engine of ACM and IEEE, filtered by year and corresponding to NS2/NS-2, NS3/NS-3, OMNeT++, QualNet and JSIM/J-SIM. March 2014.

- **Perfect Link implementation:** a Perfect Link in NS-3 is just a point-to-point link model (available in the standard NS-3 library) with a high enough throughput for obtaining a simulated transfer time equals to zero (the internal event corresponding to the receipt of the packet by the Port Node is scheduled at the same time than the sending by the linked node). In order to remove any simulation time consumption with the transfers through the Perfect Link, the delay is also set to zero, as the inter-frame gap.

- **DEVS-compliant coordinator implementation:** the NS-3 models have the possibility to redefine the default NS-3 coordinator, containing the internal event-loop of the simulator. A DEVS-compliant coordinator usable by NS-3 is provided with the MECSYCO extension for NS-3. This coordinator corresponds to the default coordinator, with a pause after each internal event processing. The model-artifact is able to request the next internal event processing. With this feature, the m-agent is able to use the model-artifact for handling the simulation synchronization and manage the external event exchanges.

Linking the library to NS-3 models enables to integrate them in a MECSYCO co-simulation.

The NS-3 software was not modified, so there is no need to recompile it and any already installed NS-3 software can be

used for executing NS-3 models integrated in a MECSYCO co-simulation.

The next section illustrates with the pseudo-codes associated to our proof of concept, using the library.

6.4 Experimentation

The MECSYCO schema corresponding to our proof of concept is described in Fig. 9 (symbols corresponds to those described in section 3.2).

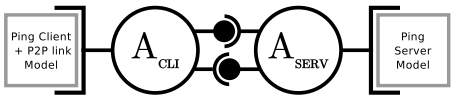

Figure 9: MECSYCO schema corresponding to the ping use-case in the two-simulators version.

The corresponding network topology is described in Fig. 10.

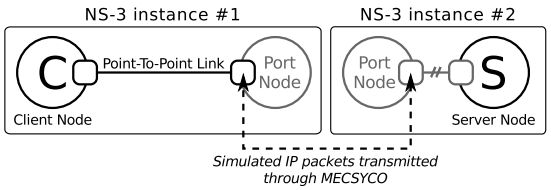

Figure 10: Network topology corresponding to the ping simulation, over two simulators thanks to Port Nodes and a Perfect Link (with two parallel bars).

The pseudo-code corresponding to the client side is given in Fig. 11. The first line corresponds to the switch from the default coordinator to the MECSYCO one. Then we instantiate two nodes, corresponding to the client node and the Port Node. The first node is enhanced with a client ping application, while the role of the Port Node is determined thanks to the registration of this node as a MECSYCO port ("P1" corresponds to the port ID). Both nodes are connected with a point-to-point link model.

```
Bind("SimulatorType", "MecsycoCoordinator")

nodes = new Nodes(2)
link = new PointToPoint
clientApp = new UdpEchoClient

netdevs = link.install(nodes)
clientApp.install(nodes.first)
MECSYCO::RegisterPort(netdevs.second, "P1")
```

Figure 11: NS-3 pseudo-code corresponding to the client side model.

The pseudo-code corresponding to the client side is given in Fig. 12. We again instantiate two nodes, this time corresponding to the server node and the Port Node. The first node is registered as a MECSYCO port, while the second one is enhanced with a server ping application. This time, the link coupling both nodes together, is a Perfect Link.

```
Bind("SimulatorType", "MecsycoCoordinator")

nodes = new Nodes(2)
link = new PerfectLink
serverApp = new UdpEchoServer

netdevs = link.install(nodes)
MECSYCO::RegisterPort(netdevs.first, "P1")
serverApp.install(nodes.second)
```

Figure 12: NS-3 pseudo-code corresponding to the server side model.

For each model, we create a MECSYCO m-agent for executing it. The pseudo-code corresponding to the MECSYCO m-agent instantiations are given in Fig. 13 for the client side and in Fig. 14 for the server side. For each, a sender and a receiver coupling-artifact are created to enable MECSYCO to exchange events from a port to another ("Cli2Serv" and "Serv2Cli" correspond to topic names for the communication middleware used by MECSYCO). This code is not specific to NS-3 and is the same for any integration in MECSYCO.

```
model = new PingClientModel
modelArtefact = new ModelArtefact(model)
agent = new Agent(modelArtefact)
sender = new CouplingArtSender("Cli2Serv")
receiver = new CouplingArtRecver("Serv2Cli")

agent.addInputCouplingArtifact(receiver, "P1")
agent.addOutputCouplingArtifact(sender, "P1")
agent.start()
```

Figure 13: MECSYCO pseudo-code corresponding to the agent instantiation for the client side.

```
model = new PingServerModel
modelArtefact = new ModelArtefact(model)
agent = new Agent(modelArtefact)
sender = new CouplingArtSender("Serv2Cli")
receiver = new CouplingArtRecver("Cli2Serv")

agent.addInputCouplingArtifact(receiver, "P1")
agent.addOutputCouplingArtifact(sender, "P1")
agent.start()
```

Figure 14: MECSYCO pseudo-code corresponding to the agent instantiation for the server side.

We also wrote the corresponding one-simulator version, to use its execution as control test for checking the simulation results. This version was presented in Fig. 7 and the corresponding pseudo-code is given in Fig. 15.

Thanks to the library, we only need to add one line in the models to switch to the DEVS-compliant coordinator, and one line by Port Node to create ports.

The results corresponding to this proof of concept and their analysis are discussed in the following section.

```
nodes = new Nodes(2)
link = new PointToPoint
clientApp = new UdpEchoClient
serverApp = new UdpEchoServer

netdes = link.install(nodes)
clientApp.install(nodes.first)
serverApp.install(nodes.second)
```

Figure 15: NS-3 pseudo-code corresponding to a model for a ping between two nodes connected by a point-to-point link.

7. DISCUSSION

The simulation time associated to each message sent or received must be exactly the same, experimentally showing that the coupling with MECSYCO has no effect on the simulation. In order to validate the relevance of the simulation results obtained via the two-simulators version thanks to our coupling with MECSYCO, we need to compare them with the simulation results obtained via the one-simulator version (purely NS-3). We know that the NS-3 submodels used are non-stochastic.

We log the simulation time of each messages sent or received, by both the client and the server in the one-simulator version, as a control test. Then we do the same measures for the client and the server, in the two-simulators version. We group the measures from the client and the server for the two-simulators version in a same log file, on a same timeline. The number of events for both versions should be identical, and the simulation time associated to each one should also be identical.

Each line of the event log file for the one-simulator version can be represented as an event contained in a series $(e_0^a, e_1^a \cdots e_{n-1}^a)$ with n the total number of lines. Each line of the log file corresponding to the two-simulators version can be represented as an event contained in a series $(e_0^b, e_1^b \cdots e_{m-1}^b)$ with m the total number of lines. As a first check, n and m must be equal.

We suppose a function $t : \mathcal{E} \to \mathbb{R}$ with $\mathcal{E}$ a set of simulation events, and associating each event with its simulated time. With the t function, we calculate the gap between the simulation times logged by the one-simulator version and the two-simulators one, as shown in the equation described in (1).

$$\sum_{i=0}^{i=n-1} |t(e_i^a) - t(e_i^b)| \qquad e_i \in \mathcal{E} \qquad (1)$$

After repeating this calculation a hundred times for a hundred different log files from different executions and summing all results, we do not found any difference. This final result experimentally shows that our two-simulators version with MECSYCO returns exactly the same simulation times than the one-simulator one (purely NS-3). As a result, our solution for multi-modeling an IP network with Port Nodes and a Perfect Link is transparent towards the simulation time.

We can now replace the point-to-point link model in our proof of concept by a PLC (Power-Line Communication) one, without modifying the server model. We could also replace the client or server side by any another simulator already integrated to MECSYCO, including other IP network simulators. For spatial couplings, either the other simulator uses IP packets in real world format like NS-3, either we use transformation operations at the coupling-artifact level in MECSYCO, for converting the representation.

This basic proof of concept allow us to multi-model bigger IP networks. Thanks to the route calculation possibilities, the client or server node could be replaced by a complex IP network modeled by NS-3, as shown in Fig. 16c. Starting from the system network, the modeler just has to split his IP network into parts (Fig. 16a) and model each part individually, then construct the multi-model thanks to the MECSYCO graph (Fig. 16b). Finally he can execute the co-simulation with MECSYCO (Fig. 16c).

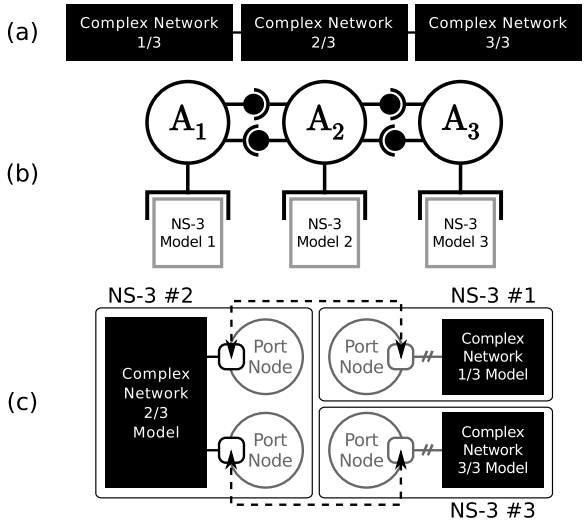

Figure 16: (a) System network splitting. (b) MECSYCO graph for constructing the multi-model. (c) Co-simulation with three NS-3 instances spatially coupled.

Thanks to the decentralized execution of MECSYCO, each model can be executed on a remote machine, for scalable simulations.

8. CONCLUSION

The goal of this paper was to present our solution for modeling a complex IP network, using models implemented for different IP network simulators, and taking into account the possibility to couple with heterogeneous simulators, for the smart grids simulations needs. We proposed a solution based on the DEVS formalism, using the MECSYCO platform.

As a proof of concept, we integrated NS-3, a popular non DEVS-compliant IP network simulator. We built a multi-model representing exchanges between a client and a server.

The client and the server was executed on independent NS-3 software instances. The MECSYCO extension for NS-3 should enable to connect a NS-3 model with any other model integrated in the MECSYCO platform, including models executed by other IP network simulators.

As future works, we plan to integrate another IP network simulator like OMNeT++, and couple an OMNeT++ model with an NS-3 one. We also plan to complete our solution with structural couplings, to connect the other heterogeneous models and simulators needed in the smart grids context, thanks to MECSYCO.

Acknowledgement

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

9. REFERENCES

[1] R. E. Bryant. Simulation on a distributed system. *Proc. of the 16th Design Automation Conference*, pages 544–552, 1979.

[2] B. Camus, C. Bourjot, and V. Chevrier. Combining DEVS with multi-agent concepts to design and simulate multi-models of complex systems (WIP). In *Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium (TMS/DEVS 15)*. Society for Computer Simulation International, TBP.

[3] K. Chandy and J. Misra. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering*, SE-5(5):440–452, Sept. 1979.

[4] J. S. Dahmann, R. M. Fujimoto, and R. M. Weatherly. The department of defense high level architecture. In *Proceedings of the 29th Conference on Winter Simulation*, WSC '97, pages 142–149, Washington, DC, USA, 1997. IEEE Computer Society.

[5] H. Farhangi. The path of the smart grid. *IEEE Power and Energy Magazine*, 8(1):18–28, 2010.

[6] R. M. Fujimoto. Parallel simulation: parallel and distributed simulation systems. In *Proc. of WSC '01*, pages 147–157. IEEE Computer Society, 2001.

[7] E. Galli, G. Cavarretta, and S. Tucci. HLA-OMNET++: An HLA compliant network simulator. In *12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications, 2008. DS-RT 2008*, pages 319–321, Oct. 2008.

[8] George Riley. PDNS - parallel/distributed NS, Mar. 2014.

[9] T. R. Henderson, M. Lacage, and G. F. Riley. Network simulations with the ns-3 simulator. In *In Sigcomm (Demo*, 2008.

[10] K. Hopkinson, X. Wang, R. Giovanini, J. Thorp, K. Birman, and D. Coury. EPOCHS: a platform for agent-based electric power and communication simulation built from commercial off-the-shelf components. *IEEE Transactions on Power Systems*, 21(2):548–558, 2006.

[11] T. Kim, M. H. Hwang, and D. Kim. DEVS/NS-2 environment: An integrated tool for efficient networks modeling and simulation. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 5(1):33–60, Jan. 2008.

[12] H. Lin. *Communication Infrastructure for the Smart Grid: A Co-Simulation Based Study on Techniques to Improve the Power Transmission System Functions with Efficient Data Networks*. PhD thesis, Oct. 2012.

[13] H. Lin, S. Sambamoorthy, S. Shukla, J. Thorp, and L. Mili. Power system and communication network co-simulation for smart grid applications. In *Innovative Smart Grid Technologies (ISGT), 2011 IEEE PES*, pages 1–6, 2011.

[14] J. Nutaro. adevs: A discrete EVent system simulator.

[15] J. Nutaro, P. Kuruganti, and L. e. a. Miller. Integrated hybrid-simulation of electric power and communications systems. In *IEEE Power Engineering Society General Meeting, 2007*, pages 1–8, 2007.

[16] J. Pelkey and G. Riley. Distributed simulation with MPI in ns-3. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, SIMUTools '11, pages 410–414, ICST, Brussels, Belgium, Belgium, 2011. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[17] G. Putrus, P. Suwanapingkarl, D. Johnston, E. Bentley, and M. Narayana. Impact of electric vehicles on power distribution networks. In *IEEE Vehicle Power and Propulsion Conference, 2009. VPPC '09*, pages 827–831, Sept. 2009.

[18] G. Quesnel, R. Duboz, and É. Ramat. The virtual laboratory environment – an operational framework for multi-modelling, simulation and analysis of complex dynamical systems. *Simulation Modelling Practice and Theory*, 17(4):641–653, Apr. 2009.

[19] A. Ricci, M. Viroli, and A. Omicini. Give agents their artifacts: The a&a approach for engineering working environments in MAS. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '07, pages 150:1–150:3, New York, NY, USA, 2007. ACM.

[20] G. F. Riley. The georgia tech network simulator. In *Proceedings of the ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network Research*, MoMeTools '03, pages 5–12, New York, NY, USA, 2003. ACM.

[21] J. Siebert, L. Ciarletta, and V. Chevrier. Agents and artefacts for multiple models co-evolution: building complex system simulation as a set of interacting models. In *Proceedings of AAMAS '10*, AAMAS '10, pages 509–516, 2010.

[22] N. US Department of Commerce. NIST framework and roadmap for smart grid interoperability standards, release 2.0. Technical report.

[23] H. Vangheluwe. DEVS as a common denominator for multi-formalism hybrid systems modelling. In *IEEE International Symposium on Computer-Aided Control System Design, 2000. CACSD 2000*, 2000.

[24] B. P. Zeigler, T. G. Kim, and H. Praehofer. *Theory of Modeling and Simulation*. Academic Press, Inc., Orlando, FL, USA, 2nd edition, 2000.