Study about decomposition and integration of continuous systems in discrete environment

Thomas Paris, Alexandre Tan, Vincent Chevrier, Laurent Ciarletta

Study about decomposition and integration of continuous systems in discrete environment

Thomas Paris
Université de Lorraine
CNRS, Inria, LORIA, UMR 7503
Vandœuvre-lès-Nancy, F-54506, France
thomas.paris@loria.fr

Alexandre Tan
Inria
LORIA, UMR 7503
Villers-lès-Nancy, 54600, France
alexandre.tan@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

Abstract

A complex system is one composed of many interacting heterogeneous entities. This kind of system can be dealt with multi-modeling and co-simulation but individual models may also be heterogeneous (continuous, discrete, event-based...). To manage this complexity, we use MECSYCO[1], a DEVS[2] compliant environment for co-simulation.

MECSYCO handles heterogeneity issues, but the number of models which may interact during a co-simulation of a complex system raises performance issues and it is important to develop performance measurement tools to study these issues with MECSYCO.

In this article we present modular performance measurement tools for MECSYCO. We exemplify the use of these tools on our "Multi-Room Heating" toy model, a scalable continuous system, to assert the tradeoff between accuracy and computational time when integrating continuous system in a discrete modeling environment. We explore the impact of decomposing a continuous system contained in one FMU[3] into several FMUs which interact. Finally we study how, under some conditions, a large model that cannot be solved on one block, can be decomposed into smaller ones, solved and simulated in a co-simulation on MECSYCO without significant loss of accuracy.

Keywords: Co-simulation; Decomposition; FMI[4]

1 Introduction

Modeling a complex system which combines discrete and continuous behaviors is not simple. More generally, it can be hard to model a complex system composed of many heterogeneous entities. One way to deal with this complexity is to use a multi-paradigm approach [12]. The complex system is decomposed into subsystems which are modeled separately, and then these models are combined to make a multi-model. The different models interact together to reproduce the behavior of the entire system. In a co-simulation, each model is associated to its own well-adapted simulator and simulators are coupled and exchange data during the simulation (simulators communicate their time and variables to update their inputs and to synchronize their execution). Then it is interesting to be able to reuse existing checked models [7]. But reusing existing models raises the heterogeneity issues.

The growing interest for co-simulation in the M&S community leads to the development of the Functional Mockup Interface [3]. This is an emerging standard which aims to ease model-exchange and co-simulation between different modeling software. FMI is an interface for time dependent model described with differential, discrete and algebraic equations. FMI for co-simulation solves the heterogeneity issues at the execution level, it describes an interface to interact with a model and its simulator. FMUs for co-simulation just need a FMI compliant modeling environment to manage the execution. Several works, such as [11, 2, 1], have been done to enable the use of the FMI standards in modeling tools, to test their FMI compliancy, or to improve the usage of FMUs.

We are working on MECSYCO [5], a modeling environment based both on the DEVS formalism [13] and Multi-Agent concepts like Agent&Artifact [10]. MECSYCO deals with heterogeneity issues and enables the interaction of models from different software and based on different formalisms in a co-simulation, it can handle FMUs for co-simulation. In this paper, we are interested in the simulation of a large number of models with many interactions. Such systems raise performance issues and we need to scale-up. Then it is particularly relevant to give performance measurement tools

[1] Multi-agent Environment for Complex-SYstem CO-simulation
[2] Discrete EVent System specification
[3] Functional Mockup Unit
[4] Functional Mockup Interface

to MECSYCO to analyze its abilities (computational time, communication time, accuracy of results etc...). The co-simulation of a complex system implies to decompose it into subsystems which interact. Since FMI is an emergent standard for co-simulation, notably for continuous systems, it is relevant to study the impact of decomposing FMUs in MECSYCO.

In this article, we present performance measurement tools on MECSYCO adapted to its Agent&Artifact architecture. After that we use these tools to collect performance indicators when we decompose a continuous system from Modelica into smaller ones contained in FMUs and when we simulate them in the discrete environment of MECSYCO. These simulations aim to exemplify the use of our tools and to evaluate both the use of FMUs with MECSYCO and the impact of the decomposition of a continuous system when we simulate it as a co-simulation of subsystems.

2 Background

This section presents the different tools used in this article.

2.1 MECSYCO

MECSYCO is a software for complex system modeling and simulation. It enables the reuse of models from different modeling tools (with their simulator) to build a multi-model, then the simulation of this multi-model is a co-simulation between models with their simulator. In order to ensure that, MECSYCO is based on the DEVS formalism [13] and on Multi-Agent concepts such as A&A (Agent & Artifact) [10].

The architecture of MECSYCO is based on the A&A concept, there are four main entities which describe our multi-models and run the simulation:

- M-agent: Model-agents are the dynamic entities of the system, they handle the simulation. Each of them is in charge of one model.
- Model-artifact: Artifacts are tools used by agents to interact with their environment. A model-artifact is used to encapsulate each model into the DEVS formalism and it makes the m-agent able to interact with it. Each m-agent is connected to one model-artifact.
- Coupling-artifact: These artifacts represent the connections between m-agents and enable data exchange between models during the simulation. Operations on time and data can be performed during the data exchange to resolve representation heterogeneity issues between models.
- Model: In MECSYCO, the models represent pre-existing models with their simulator. They come from other modeling software.

Moreover, in MECSYCO agents use the Chandy-Misra-Bryant algorithm to synchronize their simulator. It is a conservative and decentralized simulation algorithm which enables the distribution of the co-simulation through different computers.

2.2 Modelica

Modelica is an object-oriented modeling language adapted to system modeling and simulation [6]. Modelica is attractive because it is an object-oriented language which enables a modular and hierarchical construction of models. Moreover, Modelica is an equation-based modeling, hence it is particularly adapted for the design of continuous systems described by differential equation systems. We choose to use Modelica because all these properties enable the hierarchical build of model and then an easy decomposition.

2.3 FMI

FMI (Functional Mockup Interface) is an emerging standard which aims to ease model-exchange and co-simulation between different modeling tools [3]. An FMU is a model exported following the FMI standard. An FMU can be seen as a black box with some inputs and outputs, and an explicit interface to interact. An XML file describes the model (parameters, inputs, outputs etc...) and specifies if some optional C-functions are available. All the functions defined by the FMI interface are stored in a binary. All these elements are stocked in a zip file with extension ".fmu". For co-simulation, FMUs must be managed by a master software which leads the simulation assuming that data exchange between FMUs is restricted to discrete communication points.

3 Performance measurement tools

This section presents the development and the deployment of our performance measurement tools on MECSYCO.

3.1 What do we need to measure?

The first question when we want to analyze a model performance is to wonder what we need to measure. Many parameters must be taken into account to analyze a model performance, they can be classified into four groups [4]:

- Results of the model (easy to understand, accurate, good description of the system behavior)
- The validation of the model (error, accuracy, credibility)
- Resources needed (Construction time and cost, computational time and cost, result analysis time and cost, hardware requirement)
- Future use (portability, reusability)

In our case we focus on measures related to model decomposition and co-simulation. We develop tools for

data logging (to compare different execution and compute the accuracy) and for computational time logging. As MECSYCO uses a decentralized simulation algorithm, for distributed simulation we must be able to use the same tools. Consequently we need to log computational time for each model simulated. Additionally, we want to log data about the algorithm execution like the number of events and synchronizations.

3.2 Constraints

MECSYCO is an environment for complex system modeling which is still under development. Specific implementations could change and new features will be added in the future. Therefore, to be long-term useful our performance measurement tools must be code-independent and generic. Particularly, many different kinds of models may interact in a MECSYCO co-simulation, then the tools must not be dependent of the nature of model. Naturally, these tools should not disturb the simulation and the measures, especially for time analysis.

3.3 Method

3.3.1 Concept

MECSYCO uses the A&A concept, the simulation is conducted by the m-agents which use the artifacts to lead their model and to exchange data. This modular architecture is interesting to collect performance indicators because at each step, agents need to use their artifacts. That means that if we give to our artifact new abilities to log their internal functioning, we are able to get many relevant data about our co-simulation: exchanged events, computational time, the number of internal/external events for each model etc...

We choose to use the design pattern decorator which is particularly adapted to our A&A architecture. Indeed, it lets us add some features to our entities without changing their internal functioning. When we decorate an artifact, we give it new features to collect measures each time an agent uses it. The same method can be used on the agents, for instance to measure their computational time.

3.3.2 Implementation

In MECSYCO, the main concepts (agent, artifact...) are directly associated to an implementation. Our main idea is to encapsulate our pre-existing piece of code for agent and artifact into decorators which copy their behavior while adding some features. For example, we have created a model-artifact decorator (Fig. 1), it takes a model-artifact as parameter and copies its behavior by calling its functions. With this architecture we can add some operations before and after the normal model-artifact functions. A model-artifact is used to encapsulate a pre-existing model into DEVS, it implements five main functions. These functions are used for the execution of the simulation algorithm. Adding operations before and after these functions lets us get information about the algorithm such as the number of events exchanged, the computational time to process each step etc... We use the same approach on coupling artifacts to collect data about the events exchanged, and on m-agents to estimate the computational time of the algorithm.

3.3.3 Use

Finally, when we want to get some extra information about a multi-model, we just have to decorate the artifacts (or the agents) used to build it. These new artifacts will log information about events exchanged, computational time, algorithm execution etc... Then we can use them for post-mortem analysis.

To be easy to understand, we choose to create decorators for each specific task (log of times, data, execution etc...). This is not restrictive because decorators are modular and can be composed. We can easily exchange an artifact decorated with another to collect other data or we can decorate several times an artifact to use the features proposed by several tools at the same time. For now, we are able to measure:

- the total computational time for each model
- the computational time to process internal events
- the computational time to process external events
- the number of internal and external events for each model
- the number of synchronizations and exchanged events between two models

4 Generalization

Our tools let us enhance the behavior of our MECSYCO entities to collect data during the execution. But when we have to test several multi-models, it is very tiresome to change every single MECSYCO artifacts. We want to automate the exchange between different artifacts when we look for performance indicators on a multi-model. The automation of performance indicator collection raise a new issue, on MECSYCO there is no formal structure to build multi-models. Without structure we are unable to create generic functions for the MECSYCO multi-models. Therefore we must add a structure for our multi-models and this structure must be compliant with the actual properties of MECSYCO.

4.1 Structuring MECSYCO multi-model

MECSYCO is based on the DEVS formalism, each agent which is in charge of a model using a model-artifact can be considered as an atomic DEVS model. Then a MECSYCO multi-model can be considered as a set of interacting atomic DEVS models. This is helpful because DEVS defines a structure for this [13], the

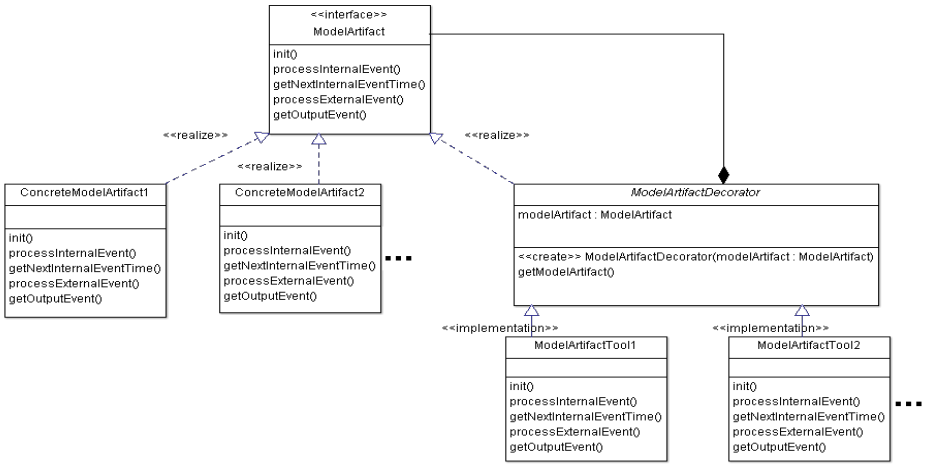

Figure 1: UML diagram of our model-artifact decorator.

DEVS coupled model structure. This structure looks like this:

$$N = (X, Y, D, \{M_d | d \in D\}, EIC, EOC, IC)$$

- D is the set which contains the names of the submodels
- X is the set of inputs
- Y is the set of outputs
- $\{M_d | d \in D\}$ is the set of DEVS submodels
- EIC (*External Input Coupling*), represents the connections between the inputs of the multi-models and the inputs of the submodels
- EOC (*External Output Coupling*), represents the connections between the outputs of the submodels and the outputs of the multi-model
- IC (*Internal Coupling*), represents the connections between the submodels

This structure defines the models used in a multi-model and the way they interact, it is easily adapted to the multi-agent description of MECSYCO multi-models. The set of submodels becomes a set of model-agents, and the set IC is adapted to take into account the coupling-artifact abilities. Indeed, in DEVS, IC is a set $\{((a, op_a), (b, ip_b) \| a, b \in D, op_a \in OutputPort_a, ip_b \in InputPort_b)\}$. However, in MECSYCO we must also define the list of operations (on time and on data) we have to perform at each exchange. Finally the structure of the MECSYCO multi-model is formalized with the set:

$$MM = (Names,\ X,\ Y,\ A,\ EIC,\ EOC,\ IC)$$

- *Names* is the set which contains the names of agents
- X is the set of input ports
- Y is the set of output ports
- A is the set of agents associated to one model thanks to a model-artifact
- EIC represents the connections between the input ports of the multi-models and the input ports of the submodels
- EOC represents the connections between the output ports of the submodels and the output ports of the multi-model
- IC=$\{\{((a, op_a), operation_{time}, operation_{data}, (b, ip_b) \| a, b \in D, op_a \in OutputPort_a, ip_b \in InputPort_b)\}\}$ represents the connections between the submodels

4.2 Implementation

We worked on the java implementation of MECSYCO, we have defined a new object multi-model which contains the adapted DEVS coupled model structure. We define an abstract class which contains our new structure and few functions to handle the multi-models (to start a simulation for instance). Now, instead of building a multi-model as a simple process, we define it as a structured object easier to handle and which enables generic analysis.

Now with our modular tools we are able to:

- put or remove performance measurement tools
- put several tools on a single entities
- perform measures on subsets of multi-models

5 Test

In this section, we exemplify the use of our performance measurement tools on a basic continuous system example using FMUs. A continuous system is a good starting point for various reasons. Continuous systems are very common in the modeling and simulation field, and there are a lot of dedicated tools (like Dymola) to simulate them. These specific tools are quite accurate and

allow to easily compute a reference solution to compare with the results of the different. We choose to run our tests with a toy example (a basic thermic system) easy to decompose and to make more complex.

5.1 Presentation of "Multi-Room Heating"

"Multi-Room Heating" is a model which represents temperature evolution in four rooms under the influence of the outside temperature, this model is closed to the one in [8]. It is a continuous system defined by two main equations. The equation

$$C * \frac{dT(t)}{dt} = Q(t)$$

represents the behavior of a room. $T(t)$ is the temperature inside the room, C the heat capacity of the volume of air and $Q(t)$ is the incoming heat flow. This incoming heat flow is determined with the equation

$$Q_i(t) = G * (T_a(t) - T_b(t))$$

which represents the behavior of a wall i. $Q(t)$ is the heat flow through the wall, G is the thermal conductance of the wall, $T_a(t)$ and $T_b(t)$ represents the temperatures at the two faces of the wall.

If we consider that the outside temperature is $T_{out}(t) = A * sin(t * f) + B$ where A is a amplitude, B an offset temperature and f a frequency adapted to have the evolution of the outside temperature in a day (86400 seconds). The problem with a single room becomes:

$$C * \frac{dT(t)}{dt} = G * (A * sin(t * f) + B - T(t))$$

This equation has an analytical solution:

$$T(t) = T(0) * e^{-\frac{G}{C} * t}$$
$$+ \frac{A * G^2 * (sin(t*f) - \frac{C*f}{G} * cos(t*f) + \frac{C*f}{G} * e^{-\frac{G}{C}*t})}{G^2 + C^2 * f^2}$$
$$- B * e^{-\frac{G}{C}*t} + B$$

This analytical solution could have been interesting but the purpose of this kind of models is not the modeling of one room but the modeling of many rooms interacting (to represents a building for instance). In most cases the analytical solution does not exist or is too hard to compute and we must compute approximate solutions. That is why we use the results of Dymola as a reference for our experiments.

This continuous system is interesting because it can be easily complexified by adding rooms and connections between these rooms. Analytical solutions become quickly complicated to find and we have to use approximate solutions. Our main example is constructed with four rooms (Fig. 2).

5.2 Simulations

We simulate our "Multi-Room Heating" using Modelica first, on Dymola, this simulation is used as a reference. Then, we export the entire model in a single

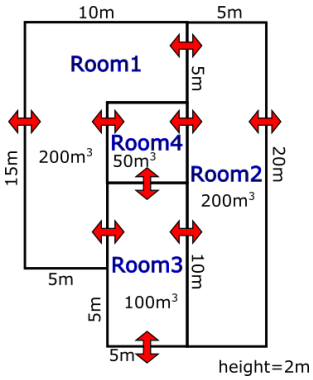

Figure 2: "Multi-Room Heating" model.

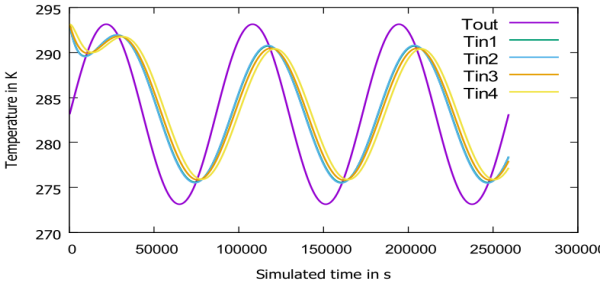

Figure 3: Evolution of the temperature of 4 rooms under the influence of an outdoor temperature during 3 days.

FMU, and finally we decompose it into several parts (rooms, walls and the outside temperature), each exported in FMUs. These FMUs are connected to rebuild the "Multi-Room Heating" and are used in a co-simulation on MECSYCO. This allows us to compare the accuracy of MECSYCO, depending on the time step size, with our reference results. Performing the same tests with a single FMU containing the entire model allow us to evaluate the impact of decomposing a continuous system into several FMUs instead of using one single FMU. We choose to export our FMUs with JModelica because it is an open-source Modelica platform. We import the FMUs in the java version of MECSYCO with a FMI model-artifact based on JavaFMI, an open-source java library. In our simulations our systems represents the evolution of the temperature of rooms during 10 days. We choose to use the same constant time step size for each FMU. We follow our multi-model structure to build our test models, and then we run several simulations with our new tools encapsulating our artifacts. These simulations let us test our performance measurement tools.

5.2.1 Results

We compare our results to a set of results from Dymola to compute the accuracy. Previous tests show that the evolution of accuracy is approximately the same for the 4 rooms, so we only display the evolution of error in the room 1. The Figure 3 shows the behavior of our system for three days.

The computational time on Dymola is not displayed

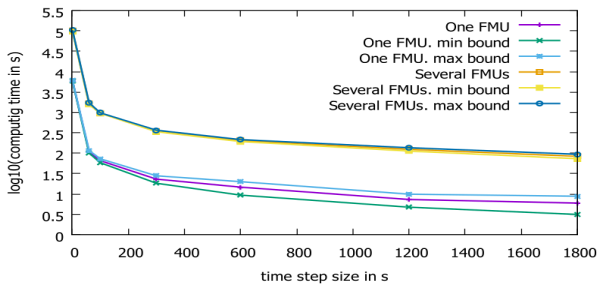

Figure 4: Evolution of the mean computational time (with min and max bounds) depending on the time step size, for our model with one FMU or several FMUs (logarithmic scale).

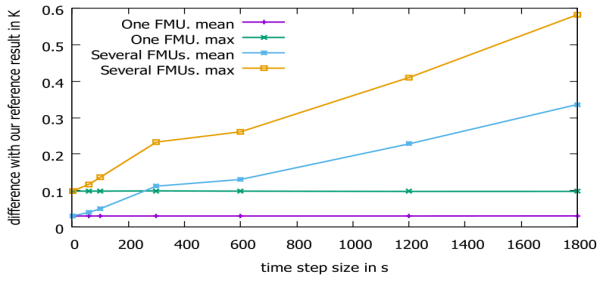

Figure 5: Evolution of the mean and max difference between our models (one FMU and several FMUs) and our reference result, depending on the time step size.

because it is very closed to the computational time obtained with one FMU on MECSYCO.

The Figure 4 shows our results concerning the computational time. In both cases (one FMU or several FMUs), the computational time is inversely proportional to the time step size. Moreover, the computational time is increased when several FMUs are used. Indeed, in our particular example, the FMUs are very simple and most of the global computing time is spent for the communications.

The Figure 5 represents the evolution of the accuracy. As expected, when using a single FMU the difference with the results of Dymola is independent of the time step size and are very similar to the results of Dymola. Our test using several FMUs shows that the difference with Dymola varies almost linearly depending on the time step size.

These results could be used to figure out a tradeoff between computing time and accuracy, but they depend on the system under study and the objectives. The tradeoff between computing time and accuracy must be determined for each system. This first experiment mainly allows to verify that the data collected by the measurement tools are consistent and validate their implementation. Then our tools can be used for further performance tests.

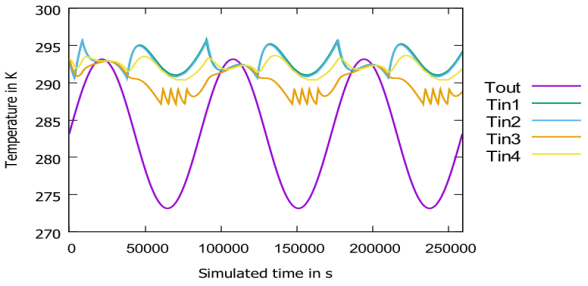

Figure 6: Graph showing the evolution of the temperature of 4 heated rooms under the influence of the outdoor temperature, view for 3 days.

5.3 Further tests

5.3.1 Adding discrete events

Our first model is a pure continuous system which evolves quite slowly. Now we want to study the impact of adding discrete events which increase our system variation. For that, we add a heater to our model on Modelica. When the temperature inside a room is lower than a value, a heater is put on and heats the room to reach quickly a maximal temperature, i.e. the temperature inside the room stays inside an interval. For this test, we choose to add an heater to rooms 1, 2 and 3 with respectively 293.15 K, 293.15 K and 288.15 K for the setpoint temperatures (see Fig. 6).

We run our test with this new feature and, as expected, we find that MECSYCO is still accurate even if the error increases more rapidly with the time step size. That shows that the tradeoff between accuracy and computational time depends closely on the model.

5.3.2 Making rooms more complex

We find that we can decompose a continuous system and simulate it using several FMUs without a significant loss of accuracy. Now we want to test that with more complex versions of our FMUs. By more complex, we mean that these new FMUs contain more equations or compute more calculation at each step. We try that according two ways: adding useless equations in our Modelica models and adding an algorithm (a matrix product) at each step. With both of these methods we run tests with an increasing number of extra computations.

In our case, the computational time increases linearly with the number of calculations per FMU, but this increase is quite low, we stay in the range of seconds. There is no impact on the accuracy, but at the end we find a limit in term of memory. We try to generate our single FMU of the entire system with JModelica first but its FMU export does not handle too many equations (about 30000) due to memory limitations. Dymola is more resilient so we use it to export our FMUs for this test. With MECSYCO as with JModelica, we find a memory limit due to the size of our FMU. In this case, the decentralized algorithm of MECSYCO combines with the possibility of decomposing our sys-

tem let us overcame this limitation by distributing our FMUs co-simulation on several computers.

5.3.3 Simulating large sets of rooms

Another way to make our system more complex is to add rooms and walls. This adds more equations and more connections in our decomposed system. We simulate our system with a varying number of rooms interconnected by walls. We just want to test our computation limits. This raises interesting questions about the build of such multi-models because it is too long to connect each model by hand. To construct these multi-models on MECSYCO we use our multi-model structure to define a parametric multi-model where we can choose the number of rooms and the way they are connected. We define a rectangular set of rooms where they are connected like a grid. We use this model to test our tools on a large set of models, thus we are able to simulate a set of about 615 FMUs and to get their computational time. We verify also that the use of our tools does not limit the number of models we can load.

6 Study of different decompositions of a system

Until now, we have compared two versions of our "Multi-room Heating" example, one where we considered it as a whole with a single FMU (Fig. 7) and the others where it is totally decomposed into 14 components (Fig. 8).

Now we study different ways to decompose our system and to figure out the best in terms of balance between accuracy and computational time. To do so, we propose five different ways to generate FMUs for the co-simulation of our system (from the whole version to the totally decomposed one). As the tools we have implemented are modular, they work whatever the configuration of the FMUs. We use them to collect the computational time and to compute the accuracy of each configuration.

6.1 First decomposition, 2 FMUs

For this first decomposition, we get out the outside temperature of our FMU (Fig. 9) and we obtain two FMUs interacting.

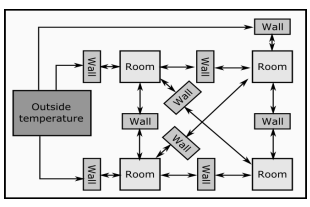

Figure 7: "Multi-Room Heating" as a single FMU.

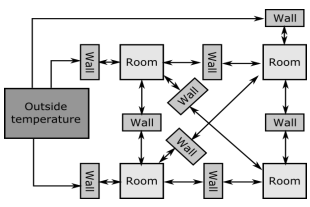

Figure 8: "Multi-Room Heating" decomposed as 14 FMUs.

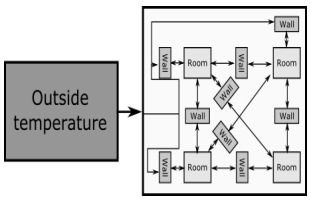

Figure 9: "Multi-Room Heating" as two FMUs.

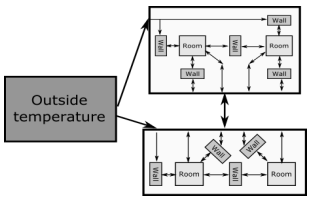

Figure 10: "Multi-Room Heating" as three FMUs.

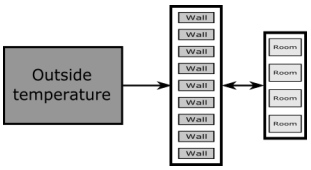

Figure 11: "Multi-Room Heating" as three FMUs, one for the outside temperature, one for rooms and the last one for walls.

6.2 Second decomposition, 3 FMUs

For the second decomposition we split our set of rooms and walls in two almost equal parts (Fig. 10). We get three FMUs.

6.3 Third decomposition, 3 FMUs

Another way to get three FMUs is to gather the elements by nature: Rooms, Wall, Outside Temperature (Fig. 11).

6.4 Study

Now we have 5 versions of the same multi-model (one FMU, 14 FMUs and the three previous possible decompositions), for each configuration we generate the appropriate FMUs and we build a MECSYCO multi-model. To compare these different configurations we choose to observe the subset of each configuration which contains the "Room 1" to log the temperature of this room at each step and to get the computing time of this FMU. We choose to collect also the total computing time of each multi-model. For each version we observe the accuracy and the computing time of a subset of the multi-model, and the computing time of the entire model. With our new multi-model structure and our decorated artifacts we build the generic tools we need and we use them on our multi-models.

6.4.1 Collected results

The Figure 12 displays computing time measures, the more there are FMUs the more we need time to compute the simulation. It is natural since all these FMUs are simple and most of the computing time is the time needed to exchange data and to synchronize the FMUs. The Figure 13 represents an evaluation of the communication time as the difference between the time spent to compute events in one FMU and the global time of the agent managing this FMU. This communication

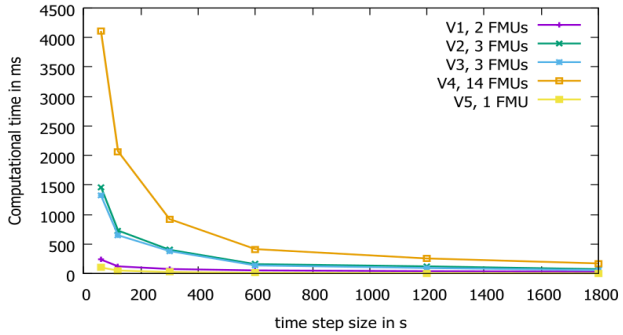

Figure 12: Evolution of the global computing time for each configuration depending on the time step size.

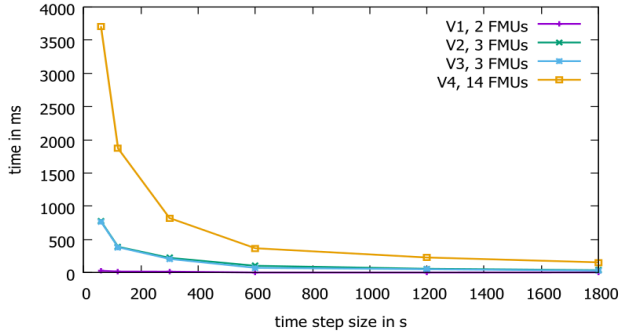

Figure 13: Evolution of the communication time for the agent which "contains" the room 1 for each configuration depending on the time step size.

time increases with the number of FMUs, in our simulations it is almost equal to the global computing time.

Finally the Figure 14 shows the evolution of the max difference of temperature between each configuration of the multi-model and the results of Dymola, depending on the time step size. This kind of result enables to determine a tradeoff between accuracy and computational time. For example, the third decomposition is not useful since it has the same evolution than the version with 14 FMUs. The second decomposition is more interesting, there is probably a compensation of errors since it is more closed to the results on Dymola with a time step of 900s than with a time step of 600s.

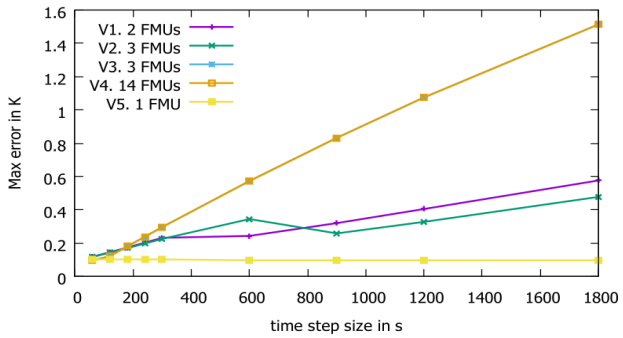

Figure 14: Evolution of the max difference of temperature with Dymola depending on the time step size.

Finally, if we consider that we are accurate enough with an error below 0.4K, the second decomposition is the best because we can keep a time step of 1200s.

7 Conclusion

We presented a set of modular performance measurement tools for MECSYCO, a co-simulation platform based both on DEVS and on Multi-Agent concepts. We applied these tools to study the impact of decomposition of a continuous system toy example to illustrate their use in the context of decomposition and integration of complex systems.

These first preliminary results are mainly focusing on solving continuous systems with multiple FMUs and on comparing their accuracy to a reference provided by the Dymola tool. Our experiments show that the use of one FMU provided results very closed to the ones of Dymola. When decomposing a complex system into several FMUs, our experiments highlight that our tools enable to study the tradeoff between accuracy and computational time when simulating several FMUs on MECSYCO. This result can be especially interesting when a single FMU can not be executed on a single machine because of memory consumption and imposes to assess different decomposition strategies.

The tools proposed in this article are not specific to continuous systems and we plan to use them on hybrid systems (with continuous and discrete models). They enable more advanced work on MECSYCO performances for potential improvements, for example a time step optimization like in [11]. An approach like in [9] could be interesting to evaluate MECSYCO on various kinds of models. This can lead also to the study of better deployment strategies for our multi-models. The structuration of MECSYCO multi-model is a first step for the integration of the concept of composition in MECSYCO, to enhance the reusability of multi-models and to enable a hierarchical build of them.

ACKNOWLEDGMENTS

We would like to thank Jean-Philippe Tavella from EDF R&D for his contribution on FMU export with Dymola.

References

[1] Christian Andersson, Johan Åkesson, Claus Führer, and Magnus Gäfvert. Import and export of functional mock-up units in jmodelica. org. In *8th International Modelica Conference 2011*. Modelica Association, 2011.

[2] Christian Bertsch and Elmar Ahle Ulrich Schulmeister. The functional mockup interface-seen from an industrial perspective. In *10 th International Modelica Conference, Lund, Sweden*, 2014.

[3] Torsten Blochwitz, M Otter, M Arnold, C Bausch, C Clauß, H Elmqvist, A Junghanns, J Mauss, M Monteiro, T Neidhold, et al. The functional mockup interface for tool independent exchange of simulation models. In *8th International Modelica Conference, Dresden*, pages 20–22, 2011.

[4] ROGER J Brooks and ANDREW M Tobias. Choosing the best model: Level of detail, complexity, and model performance. *Mathematical and computer modelling*, 24(4):1–14, 1996.

[5] Benjamin Camus, Christine Bourjot, and Vincent 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, 2015.

[6] Peter Fritzson and Vadim Engelson. Modelica - A unified object-oriented language for system modeling and simulation. In *ECOOP'98-Object-Oriented Programming*, pages 67–90. Springer, 1998.

[7] Javier Gil-Quijano, Thomas Louail, and Guillaume Hutzler. From biological to urban cells: lessons from three multilevel agent-based models. In *Principles and Practice of Multi-Agent Systems*, pages 620–635. Springer, 2012.

[8] Leilani Gilpin, Laurent Ciarletta, Yannick Presse, Vincent Chevrier, and Virginie Galtier. Co-simulation solutions using aa4mm-fmi applied to smart space heating models. In *Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques*, pages 153–159. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014.

[9] Ezequiel Glinsky and Gabriel Wainer. Devstone: a benchmarking technique for studying performance of devs modeling and simulation environments. In *Distributed Simulation and Real-Time Applications, 2005. DS-RT 2005 Proceedings. Ninth IEEE International Symposium on*, pages 265–272. IEEE, 2005.

[10] Alessandro Ricci, Mirko Viroli, and Andrea 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*, page 150. ACM, 2007.

[11] Tom Schierz, Martin Arnold, and Christoph Clauß. Co-simulation with communication step size control in an fmi compatible master algorithm. In *9th Int. Modelica Conf., Munich, Germany*, pages 205–214, 2012.

[12] Hans Vangheluwe, Juan De Lara, and Pieter J Mosterman. An introduction to multi-paradigm modelling and simulation. In *Proceedings of the AIS'2002 conference (AI, Simulation and Planning in High Autonomy Systems), Lisboa, Portugal*, pages 9–20, 2002.

[13] Bernard P. Zeigler, Herbert Praehofer, and Tag Gon Kim. *Theory of modeling and simulation : integration discrete event and continuous complex dynamic systems*. Academic press, 2000.