Study about decomposition and integration of continuous systems in discrete environment

Thomas Paris, Tan Alexandre, 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 (Multi-agent Environment for Complex-SYstem CO-simulation) a DEVS 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 also performance issues. So it's important to develop performance measurement tools to study MECSYCO's co-simulation performances.

In this article we present modular performance measurement tools for MECSYCO. We test these tools on our "Multi-Room Heating" model, a scalable continuous system, to assert the tradeoff between accuracy and computational time when integrating continuous system in a discrete modeling environment. Then we study the impact of decomposing a continuous system contained in one FMU into several FMUs which interact. We verify the validity of our tools and we show that, 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

1 Introduction

Modeling a complex system which combines discrete and continuous behavior isn't simple. More generally, it's 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 [11]. 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. Then it's interesting to be able to reuse existing checked models [7] but with this approach we have to deal with models' heterogeneity issues.

The growing interest for co-simulation in the M&S community leads to the development of the Functional Mockup Interface (FMI) [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 [10, 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're working on MECSYCO [5], a modeling environment based both on the DEVS formalism [12] and Multi-Agent concepts like Agent&Artifact [9]. 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. Even if MECSYCO handles complex system heterogeneity issues, there are still many issues for the modeling of complex systems. We may have to simulate a large number of models with many interactions. Such systems raise performance issues and we need to scale-up. So it's particularly relevant to give performance measurement tools to MECSYCO to analyze its abilities. MECSYCO aims to integrate many kinds of models, not just FMUs,

so our measurement tools must be generic enough. The co-simulation of a complex system implies to decompose it into subsystems which interact, so we must be able to study the synchronization algorithm. For large systems, we need to distribute de calculation, so we need to get measure to analyze the deployment strategy. To test our tools, we choose to study a continuous system easy to decompose because it's easy to check the validity of our results with continuous models. Since FMI is an emergent standard for co-simulation, notably for continuous systems, it's relevant to study the impact of decomposing FMUs in MECSYCO.

In this paper, we present a 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 test our tools and to evaluate both the use of FMUs with MECSYCO and the impact of the decomposition of a 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 (Discrete EVent System specification) formalism [12] and on Multi-Agent concepts such as A&A (Agent & Artifact) [9].

MECSYCO's architecture is based on the A&A concept, we have four main entities which describe our multi-models and run the simulation:

- M-agents: Model-agents are the dynamic entities of the system, they handle the simulation. Each of them is in charge of one model.
- Model-artifact: An artifact is a tool used by the agent to interact with its environment. A model-artifact is used to encapsulate each model into the DEVS formalism and it makes the m-agent able to interact with them. 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's a conservative and decentralized simulation algorithm. So we can distribute 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's an object-oriented language which enables a modular and hierarchical construction of models. Moreover, Modelica is an equation-based modeling, hence it's particularly adapted for the design of continuous systems described by differential equation systems. We choose to use Modelica for all these properties which enable the hierarchical build of a model, so it's easy to decompose.

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 (Functional Mockup Unit) 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 precises 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 manage by a master software which leads the simulation assuming that data exchange between FMUs is restricted to discrete communication points. The FMI standard exists in two versions (1.0 and 2.0), but these two versions aren't supported by every modeling tools yet.

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)

We focus on measures linked to the simulation itself, so we decide to develop tools for data logging (to compare different execution and compute the accuracy), for computational time logging. MECSYCO uses a decentralized simulation algorithm, for distributed simulation we must be able to use the same tools, so 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 a modeling environment for complex system modeling which is still under development. So 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, so the tools mustn't be dependent of the model's nature. Naturally, these tools shouldn't 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're 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've created a model-artifact decorator (Figure 1), it takes a model-artifact as parameter and copies its behavior by calling its functions. But with this architecture we can add some operations before and after the normal model-artifact's 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. So 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. The same method is used on m-agent 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:

- total computational time for each model
- computational time to process internal events
- computational time to process external events
- the number of internal and external events for each model
- 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's very tiresome to change every single MECSYCO artifacts. So, 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.

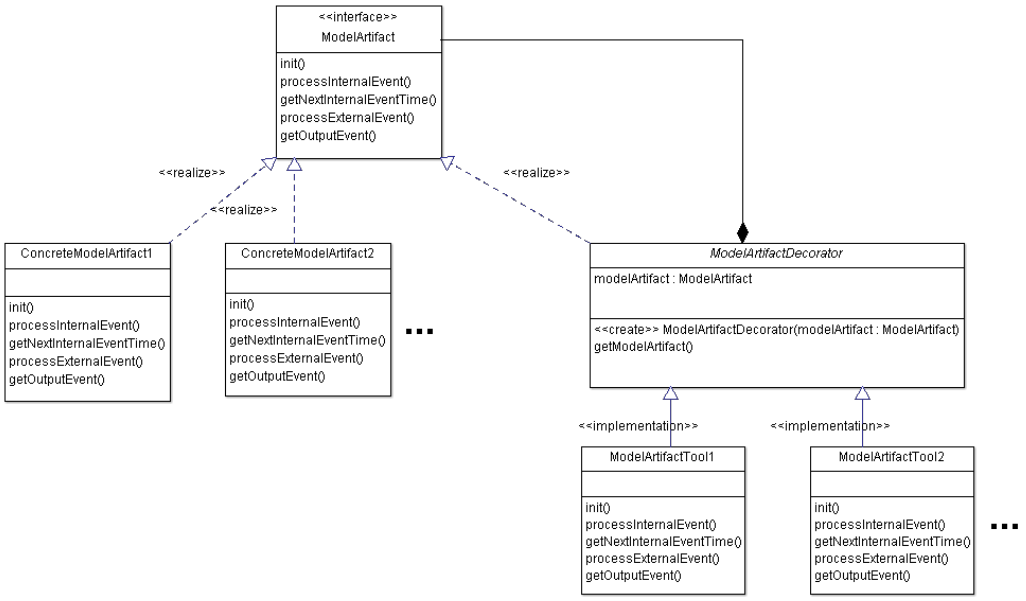

Figure 1: UML diagram of our model-artifact decorator.

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 [12], the 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 of submodels' names
- X is the set of input ports
- Y is the set of output ports
- $\{M_d | d \in D\}$ is the set of DEVS submodels
- EIC (*External Input Coupling*), represents the connections between the input ports of the multi-models and the input ports of the submodels
- EOC (*External Output Coupling*), represents the connections between the output ports of the sub-models and the output ports 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 $\{((a, op_a), (b, op_b) \| a, b \in D, op_a \in OutputPort_a, op_b \in InputPort_b)\}$. 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 of agents' names
- 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=$\{((a, op_a), op_{time}, op_{data}, (b, op_b) \| a, b \in D, op_a \in OutputPort_a, op_b \in InputPort_b)\}$ represents the connections between the submodels

4.2 Implementation

We worked on the java implementation of MECSYCO, we've defined a new object multi-model which contains the adapted DEVS coupled model structure. So 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.

So, 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

Now we want to test our new tools to evaluate the performances of MECSYCO. As the FMI is a growing standard for co-simulation, it's relevant to study our ability to import FMUs for co-simulation. 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 to simulate them. These specific tools are quite accurate so we can easily compute a reference solution to compare our results with. We choose to run our test with a basic thermic system.

5.1 Presentation of "Multi-Room Heating"

"Multi-Room Heating" is a model which represents the evolution of the temperature in four rooms under the influence of the outside temperature, this model is closed to the one in [8]. It's 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 or 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 isn't the modeling of one room but the modeling of many rooms interacting (to represents a building for instance). In most cases the analytical solution doesn't exist or is too hard to compute so we must compute approximate solutions. That's why we use Dymola's results as a reference for our experiments.

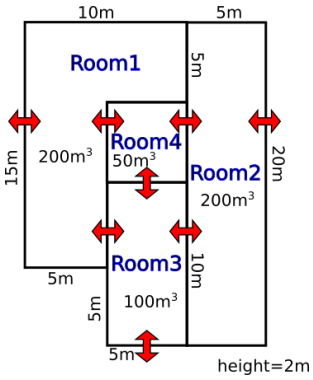

Figure 2: "Multi-Room Heating" model

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've to use approximate solutions. Our main example is constructed with four rooms (Figure 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 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's 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 to a set of results from Dymola to compute the accuracy. Previous test show that the evolution

of accuracy is quite the same for the 4 rooms, so only display the evolution of error in the room 1. The figure 3 shows the behavior of our system for three days.

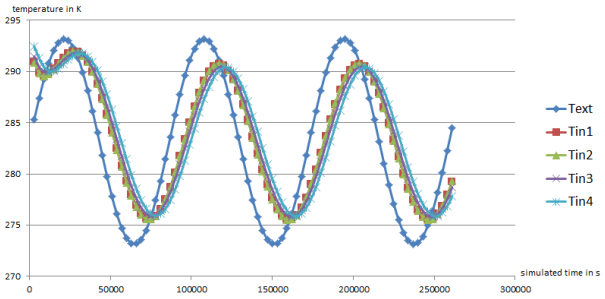

Figure 3: Evolution of the temperature of 4 rooms under the influence of an outdoor temperature during 3 days

We find that the computational time on one FMU is very closed to the one on Dymola so we don't display the computational time of Dymola.

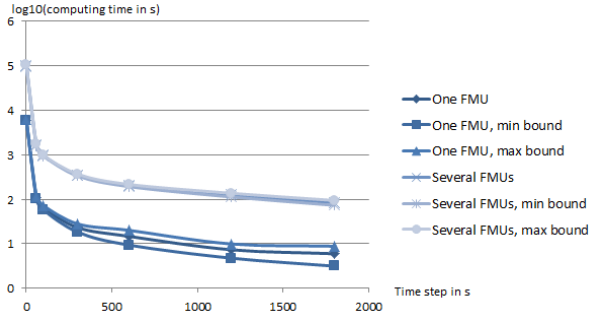

Figure 4: Evolution of the mean computational time (with min and max bounds) depending on the time step size, for our model with one FMU or several FMUs (logarithmic scale)

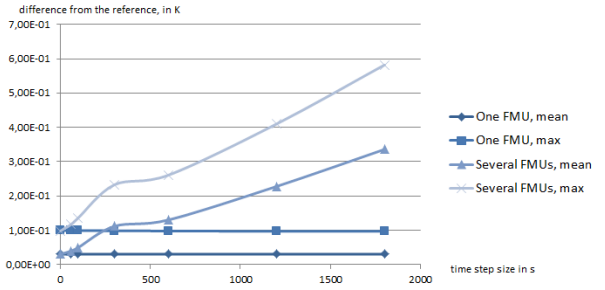

Figure 5: Evolution of the mean and max difference between our models (one FMU and several FMUs) and our reference result, depending on the time step size

This first test (Fig 4 and Fig 5) shows four main results:

- Using one FMU alone, the error is independent of the time step size. It's natural since our FMU doesn't receive any events.
- Using several FMUs, the error varies almost linearly depending on the time step size.
- Using one or several FMUs, the computational time is inversely proportional to the time step size.
- Using several FMUs increases the computational time and decrease the accuracy, this is due to the communication.

Our results highlight the impact of decomposing an FMU to simulate it on MECSYCO. Decomposing a system to simulate it with several FMUs in a co-simulation implies to add a new layer for the simulation. At each step, FMUs executes themselves with their own algorithms (typically CVode, explicit Euler etc...) but they just compute one part of the whole system, and this is the master algorithm which manages the time between each data exchange. Naturally, the time between each data exchange impact the accuracy of the simulation. That's what we try to quantify for the algorithm of MECSYCO.

Our test shows that we can keep a good accuracy in MECSYCO with several FMUs. The version with multiple FMUs is slower than the version with a single FMU, it's normal since we add extra computation for the synchronization and the exchange of data. Nevertheless for our thermic system, we can consider that error is tolerable under 0.2 Kelvin degrees so we can choose a time step size which keeps a good accuracy and a short computational time. This result depends on the system under study, the tradeoff between computing time and accuracy must be determined for each system.

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 heat 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 version of our FMUs. By more complex, we mean that these new FMUs contain more equations or compute more calculation at each step. To do that,

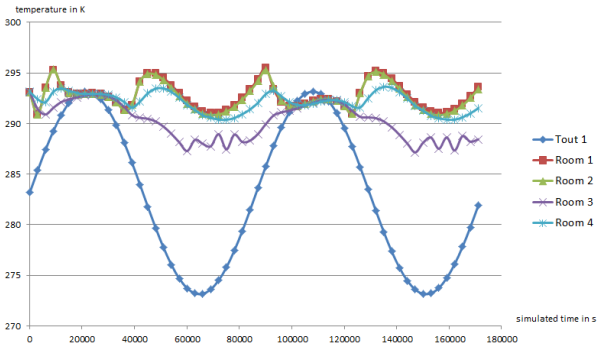

Figure 6: Graph showing the evolution of the temperature of 4 heated rooms under the influence of the outdoor temperature, view for 2 days

we try 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.

We find that the computational time doesn't increase a lot, 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 doesn't 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 system 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 for our decomposed system. So we try to simulate our system with a varying number of rooms interconnected by walls. We don't want to simulate a realistic system but just to test our computation limits. This raises interesting questions about the build of such multi-models because it's 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're 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 doesn't limit the number of models we can load.

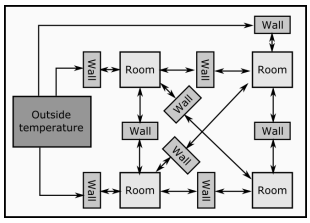

Figure 7: "Multi-Room Heating" as a single FMU

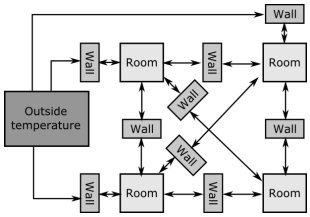

Figure 8: "Multi-Room Heating" decomposed as 14 FMUs

6 Study of different decompositions of a system

For now, we've compared two versions of our "Multi-room Heating" example, one where we considered is as a whole with a single FMU (Fig 7) and the other where it's totally decomposed into 14 components (Fig 8). Between this two views, we can find many different possible decompositions of this system.

Now we want to study different ways to decompose our system and to figure out the best in terms of 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've implemented are modular, they work whatever the configuration of the FMUs, so we use them to collect the computational time and to compute the accuracy of each configuration.

6.1 1st decomposition, 2 FMUs

For this first decomposition, we get out the outside temperature of our FMU (Fig 3 9) so we have two FMUs interacting.

6.2 2nd decomposition, 3 FMUs

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

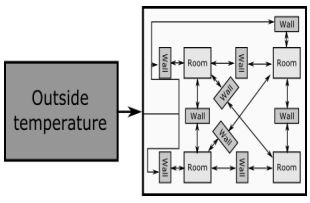

Figure 9: "Multi-Room Heating" as two FMUs

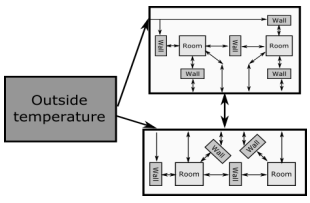

Figure 10: "Multi-Room Heating" as three FMUs

6.3 3rd decomposition, 3 FMUs

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

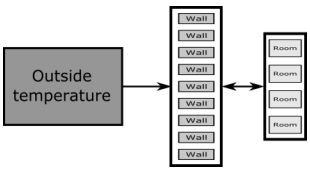

Figure 11: "Multi-Room Heating" as three FMUs, one for the outside temperature, one for rooms and the last one for walls

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. So 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 decoracted artifacts we build the generic tools we need and we use them on our multi-models.

6.4.1 Collected results

The following graphs show the results we've obtained (Fig 12, 13, 14). The first graph shows computing time measures, the more there are FMUs the more we need time to compute the simulation. It's natural since all these FMUs are light and most of the computing time is the time needed to exchange data between FMUs. The second graph (Fig 13) confirms that, it represents the difference between the time spent to compute events in one FMU and the global time, it's an evaluation of the communication time. This communication time increases with the number of FMUs, in our simulations it's almost equal to the global computing time.

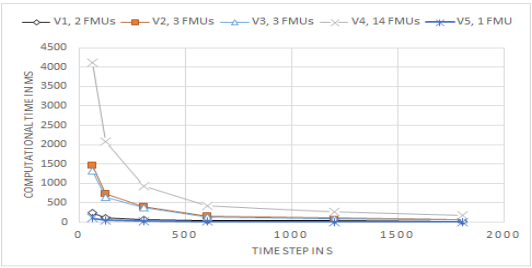

Figure 12: Evolution of the global computing time for each configuration depending on the time step

The third graph (Fig 14) shows the impact of the configurations on the max difference with Dymola's results, depending on the time step. The third decomposition isn't useful since it has the same evolution than the version with 14 FMUs. The second decomposition is more interesting, there is probably an compensation

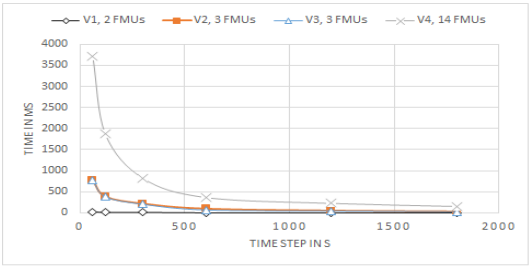

Figure 13: Evolution of the difference between the global computing time and the time used to compute internal and external events for the FMU which "contains" the room 1

of errors since it's more closed to Dymola's results with a time step of 900s than with a time step of 600s.

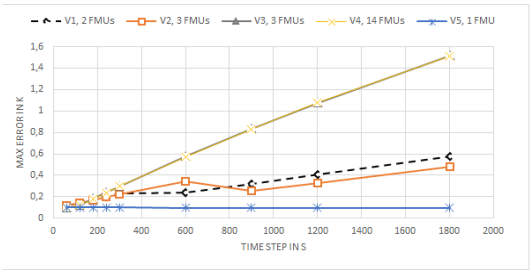

Figure 14: Evolution of the max difference between each version of our multi-model and Dymola depending on the time step

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 our first results about decomposition and integration of complex systems. This is seen from both a modeling and a simulation runtime performance issue. We've presented our methodology as well as the benchmarks and tools we 've developed. They are integrated within our MECSYCO toolset and are used to validate our approach. These preliminary results are mainly focusing on multiple FMUs (solving continuous systems) but we are also integrating those with discrete events simulators.

Our experiments show several things about the integration of FMUs in MECSYCO. We show that, as expected, simulating one FMU doesn't generate errors and we highlight the tradeoff between accuracy and computational time when simulating several FMUs on MECSYCO. Moreover, we find that the first limit when simulating complex FMUs is the memory consumption during the FMU import. This limit can be overcome with MECSYCO by decomposing the complex FMU into several lighter FMUs (if possible) to distribute the simulation. Furthermore, we know with our previous tests that we can keep a good accuracy. Some results of these tests were expected; hence they validate the im-

plementation of our tools. The last experiment shows the use of our tools to study different decompositons of a model to figure out the most interesting.

The developments done for this article may lead to further works on MECSYCO. The tools for performance measurement enable more advanced work on MECSYCO performances for potential improvements, for example a time step optimization like in [10]. 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.

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] 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.

[10] 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.

[11] 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.

[12] 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.