WIP on a Coordination Language to Automate the Generation of Co-Simulations

Giovanni Liboni, Julien Deantoni

WIP on a Coordination Language to Automate the Generation of Co-Simulations

Giovanni Liboni,*† Julien Deantoni†

*Safran Tech, Modeling & Simulation, Rue des Jeunes Bois, Châteaufort, 78114 Magny-Les-Hameaux, France
giovanni.liboni@safrangroup.com
†Université Côte d'Azur, I3S/INRIA Kairos, France
julien.deantoni@univ-cotedazur.fr

***Abstract*—System Engineering involves several disciplines to design, develop and verify complex systems, using different modeling languages with different semantics. A simulation of the global behavior from the heterogeneous executable models is used to verify and validate the emerging behavior of the system. Co-simulation is a way to realize this simulation but it requires coordinating the different heterogeneous artifacts. This coordination is not a trivial task due to the increasing complexity and heterogeneity. In this paper, we propose a language that enables the specification of a coordination between models and the automatic generation of a dedicated coordinator (Master Algorithm) with respect to the coordination and the behavioral semantics of the executable models.**

***Index Terms*—Coordination, Co-Simulation, Master Algorithm, Language Engineering**

I. Introduction

With the increasing complexity of modern software intensive systems, their development usually involves several stakeholders that focus on a specific aspect of the system. These stakeholders use dedicated languages, tailored to their domain both syntactically and semantically. This decomposition of the system specification into different domains is effective and allows domain experts to focus on their domain of expertise. It is however required to tame the emerging behavior between the artifacts from the different domains to verify and validate the whole system [1].

A partial solution to this problem is to use co-simulation, where the different executable artifacts from different domains are simulated independently, but with a coordinator in charge of keeping the data and time consistent across the different executable artifacts. For instance, the FMI standard [2], nowadays implemented by more than a hundred industrial tools, proposed to bundle, in a black box manner, the executable artifacts so that they all implement the same simulation interface. It is then the responsibility of an integrator to write the coordinator (name *Master Algorithm* in FMI) so that the global co-simulation correctly reflects the system behavior.

Work partially founded by the GLOSE bilateral project between INRIA and Safran (https://gemoc.org/glose). This work is also a follow up of interesting discussions during the CAMPAM 2017 and CAMPAM 2019 workshops.

978-1-7281-4113-8/19/$31.00 ©2019 IEEE

The problem is that writing a coordinator is far from easy and can lead to false simulation results if not realized correctly [3]–[9]. More precisely, there is a need to know, at least partially, about the behavior of the bundled executable models in order to implement a specific coordination [5], [8], [9]. This can lead to complex coordinators, emerging from the different interactions between the different executable artifacts. Some approaches like [10], [11] tried to provide dedicated language to ease this task but the expressiveness is very limited and considers almost only the topology of executable models together with their internal causalities and delay.

In order to facilitate the writing of correct coordinators, we propose to define a language dedicated to the specification of the interactions between different executable artifacts. More precisely, inspired by works on the Architecture Description Languages [12], [13], work on Coordination Languages [14] and work on heterogeneous frameworks [15]–[18], we propose to define a Model Behavioral Interface that exhibits enough information to ensure that the interaction between different models can be specified correctly (it greyfies the black box view proposed by the FMI standard so that a correct coordination can be defined). We also propose a structured dedicated language to specify the different interactions between the executable models. From such description, it is possible to automatically generate an appropriate coordinator. This work is still in progress and in this short paper we show the preliminary results we obtained.

The paper is structured as follows: the next section presents some background on co-simulation to better understand the proposition, which is detailed in Section III. Then a set of examples are presented before concluding.

II. Background

The FMI standard proposes to bundle a model together with its interpreters (being a solver or anything else) behind a time driven interface that homogenizes all the executable models. Alternatively, the HLA standard [19] proposes the same idea but using an event-driven interface. Mappings between the data from different models are then established and a *coordinator* is in charge of keeping time and data consistency between the different models under execution. Several works have shown

that neither time driven nor event driven can correctly handle all model executions. To caricature the idea, *cyber* models, where data are usually piece wise constant, fit better with a event triggered approach while *physical* models, where data are (piece-wise) continuous, fit better with a time-triggered approach[1]. Orthogonally, other studies have shown that the correctness of the co-simulation does not only rely on the correctness of each executable models but also depends on the coordinator. All this put together, it appears that implementing the coordinator is more and more complex because it must take into account (1) the characteristics of the data it conveys, (2) the characteristics of the models and their solvers and (3) the topology between the different interconnected executable models. These aspects are often neglected but it has been shown to actually condition the correctness of the co-simulation.

III. Approach

A. Overview

In this paper we give the first results of our ongoing work towards the definition of a language dedicated to the setup of co-simulation. It is based on a *Model Behavioral Interface* that exhibits the characteristics of the models and their exposed variables. Based on such interface, it is possible to define *connectors* between the data from different models. The behavior of such connectors (*i.e.*, the glue) is precisely defined and structured according to a new dedicated language. The information from the setup of the co-simulation is then used to automatically generate a coordinator.

B. Definition of the Model Behavioral Interface

The Model Behavioral Interface must provide enough information to ensure that a system integrator has the knowledge required to correctly coordinate the different executable models. Such information has been partially studied in [20]. In our work, the interface defines the type and nature of the exposed data, where the nature can be defined as *Spurious*, *Piecewise-Constant*, *Continuous*, *Piecewise-Continuous* and *Constant*. By using such information, the designer can decide the way to coordinate different data. For instance, the way to coordinate *Spurious* data (*e.g.*, ported by an event) is handled differently than the way to coordinate *Continuous* data (that may be sampled periodically). Other information is added to the data according to their nature. For instance, for *Piecewise-Continuous* input data, an annotation is added to specify if internally, the executable model is using interpolation or extrapolation (like explained in [9]). Another example of such annotations are *events* associated to *Piecewise-Constant* data, like for instance the $update$ event represents the instant at which a data is assigned or the $readyToRead$ event represents the instant before the data is actually read[2]. Inspired by the time model of MARTE [22], the Model Behavioral Interface also defines the temporal reference of the model, *i.e.*, the way time is encoded and the dimension it refers to (typically the physical time but possibly other dimensions like an angle). Finally, another example of information carried out by the model interface is the capability of the executable model to rollback or not. Such information is relevant to choose an order in the execution of the different models.

[1]Of course more generally both cyber and physical models can benefit of a mixed approach

[2]This is directly inspired from [5], [21]

C. Definition of the Coordination Model

Based on the Model Behavioral Interface, the coordination model is a set of connectors. Each connector is in charge of defining *when* and *how* one or more data are conveyed from a model to another. In order to specify it in a way which is amenable to the generation of a coordinator, we structured a coordinator through three different elements further explained in the remainder of this section: *triggering condition*, *synchronization constraint* and *interaction*.

a) triggering condition: It defines the *instant* at which the interaction (defined later) must be realized. In order to specify such constraint, we specify a logical clock as defined in CCSL [23]. A logical clock is an ordered set of instants and can represent either a clock in the traditional sense (*e.g.*, every 5ms) or any event (*e.g.*, a data update), or composition of events (*e.g.*, union of events or their intersection).

b) timing constraint: It specifies a relation between the temporal references in different models. This relation defines when the actual interaction can be done. Typically, such constraint specifies that the time must be equal when two models exchange some data. However, on one hand, we want to be more expressive, for instance to encode some time relativity effects between different timebases (for instance between the time on earth and in a satellite) or to support polychronous systems [24], *i.e.*, system whose temporal referential can come from different dimensions (*e.g.*, a distance or an angle). On the other hand, it is also important to be able to define the exact condition under which time is considered to be the same in two different models. For instance, if time is encoded as a Float in one model and as an Integer in the other. Such constraint, when respected, ensures a consistent temporal state of the two models, allowing the interaction to occur.

c) Interaction: It specifies how the data are actually exchanged between the models. It can be a simple assignment but can also contain conversions, data manipulation or even data savings like for instance when using a FIFO. It is not clear yet the expressiveness required here.

D. Generation of the Master Algorithm

We generate the Master Algorithm based on a set of model interfaces and connectors. The generation of the Master Algorithm is based on an extended version of the FMI APIs proposed in [5] and slightly improved to allow using a stop condition as a parameter for the *doStep* method. The generation is divided into three main steps. At each step, the previously identified elements are exploited and used to generate the dedicated Master Algorithm.

The first step takes into account the *Model Behavioral Interface* of the executable models. The defined properties (*e.g.*, rollback capability), the exposed variables and their nature are

exploited to define a first execution schedule. For example, a model supporting rollback receives a higher execution priority, due to the possibility to repeat the step with a smaller step size in case of step rejection from other models. Furthermore, it is possible to classify a model as a *Cyber* component if it exposes only *Piecewise-Constant* or *Spurious* variables. Then, it can be use as a temporal reference for the simulation.

During the second step, the *triggering condition* of each connector is evaluated. Then, the conditions of all connectors associated with the exposed data of the same model are collected and put in disjunction. The resulting set is used as an additional parameter for the *doStep* method of the model. In other words, the *doStep* method is parametrized with the disjunction of all the triggering conditions of the associated connectors, so that it pauses the execution as soon as the internal state of the model satisfies one condition.

The third step creates a sequence of conditional statements after each *doStep* call, which ensures timing synchronization and data propagation. These statements are generated relying on three elements: (1) *triggering conditions*, which are used in a conditional statement, which guards the code of the interaction, (2) *timing constraint*, which is used to make consistent the time in the different models according to the timing constraint, (3) the specification of the *interactions*, to generate the corresponding data propagation methods, *i.e.*, to bind the exposed variables together and set the correct typed function to use for the data propagation.

IV. Examples

We present two simple examples to illustrate the proposed language and the corresponding generated Master Algorithm.

(a) (b)

Fig. 1: Simple examples to present our approach

```
load A, B

when every 5 ms
sync A.time = B.time
do
    A.v -> B.v

when every 3 ms
sync B.time = A.time
do
    B.w -> A.w
```

Listing 1: Coordination Model for Figure 1a

Fig. 1a shows an example representing two coupled models. The Model Behavioral Interface of each model exposes two *Continuous* variables exchanging data at two different discretization rates. The coordination model specification in Listing 1 defines the multi-rate discretization using two different connectors. Each connector specifies its own rate (line 3 and 8) and the variables concerned by the coordination (line 6 and 11). During the generation of the Master Algorithm, the two different sampling rates are taken into account to compute the Greatest Common Divider. The two step sizes Δt_A and Δt_B are respectively set to 5 ms and 3 ms, and the GCD is computed and used as a parameter for the *doStep(Δt)* function. If one of the models exposes, as its property, the rollback capability, it can be exploited by the generator to choose the execution order between the models (line 6 and 7). The condition with the modulo operation (line 8 and 12) satisfies the *timing constraint* and the get/set methods (line 9 and 10) realize the *interaction*.

Algorithm 1 Pseudo code for the Master Algorithm generated from the Coordination Model Specification in Listing 1

```
1:  A := newModelA;
2:  B := newModelB;
3:  Δt := ComputeGreatestCommonDivisor(Δt_A, Δt_B)
4:  t := 0
5:  while t ≤ t_end do
6:      A.doStep(Δt)
7:      B.doStep(Δt)
8:      if t mod Δt_A then
9:          fmi2GetReal(A, v)
10:         fmi2SetReal(B, v)
11:     end if
12:     if t mod Δt_B then
13:         fmi2GetReal(B, w)
14:         fmi2SetReal(A, w)
15:     end if
16:     t = t + Δt
17: end while
```

```
load Controller as C, Plant as P

when C.p.updated
sync C.time = P.time
do
  C.p -> P.p

when C.s.readyToRead
sync P.time = C.time
do
  P.s -> C.s
```

Listing 2: Coordination Model for Figure 1b

Fig. 1b shows a system composed by a controller and a plant. The controller has an output variable *p* which is *Piecewise-Constant* and an input variable *s* which is *Continuous*. The coordination model in Listing 2 specifies that the data between the two models has to be propagated only when the value of *p* is updated and when *s* is read for internal computations. It defines the connection and the direction for the two variables (line 6 and 11), the *triggering condition* (line 3 and 8) and the *timing constraint* (line 4 and 9). Algorithm 2 illustrates the generated master algorithm and the *doStep(...)* function parametrized by the disjunction of the specified *triggering conditions* (line 5). It allows to analyze the incoming $updated$ or $readyToRead$ events, to ensure the specified timing synchronization (line 6 and 10) and to apply the *interactions* (line 8-9 and 12-13).

Algorithm 2 Pseudo code for the Master Algorithm generated from the Coordination Model Specification in Listing 2

```
C := newController;
P := newPlant;
t := 0
while t ≤ t_end do
    t_C := C.doStep(C.p.updated ∨ C.s.readyToRead)
    if C.p.updated then
        t_P := P.doStep(t, t_C − t)
        fmi2GetReal(C, p)
        fmi2SetReal(P, p)
    else if C.s.read then
        t_P := P.doStep(t, t_C − t)
        fmi2GetReal(P, s)
        fmi2SetReal(C, s)
    end if
    t := t_C
end while
```

V. Conclusion

In this paper we presented ongoing work towards the definition of a language dedicated to the specification of co-simulation setup. It is based on an expressive Model Behavioral Interface, which provides the information required by a designer to correctly define the connections between different executable models. We also provide means to specify the behavior of connectors between different data exposed by the models. This behavior is specified by three elements: a *triggering condition* to specify when interaction must be done, a *timing constraint* to specify the conditions on model time bases to allow interaction and the definition of the *interaction* itself. Based on such information we are currently defining a way to generate a coordinator based on an extension of FMI previously realized. More research needs to be completed in order to make this work in progress more concrete: we need to ensure there is a systematic way to define the coordinator from a coordination model; we need to add more properties into the Model Behavioral Interface to handle more cases (*e.g.*, delays or not for the detection of algebraic loop). Finally, we also plan to propose the definition of coordination pattern between language behavioral interface to enable the automatic generation of coordination model based on "good practice" (inspired by [17]).

References

[1] B. Combemale, J. Deantoni, B. Baudry, R. B. France, J. Jzquel, and J. Gray, "Globalizing modeling languages," *Computer*, vol. 47, no. 6, pp. 68–71, June 2014.

[2] Modelisar, "FMI for Model Exchange and Co-Simulation," July 2014. [Online]. Available: https://fmi-standard.org/downloads#version2

[3] C. Gomes, B. Meyers, J. Denil, C. Thule, K. Lausdahl, H. Vangheluwe, and P. D. Meulenaere, "Semantic adaptation for fmi co-simulation with hierarchical simulators," *SIMULATION*, vol. 95, no. 3, pp. 241–269, 2019. [Online]. Available: https://doi.org/10.1177/0037549718759775

[4] C. Thule, C. Gomes, J. Deantoni, P. G. Larsen, J. Brauer, and H. Vangheluwe, "Towards the Verification of Hybrid Co-simulation Algorithms," in *Workshop on Formal Co-Simulation of Cyber-Physical Systems (SEFM satellite)*, Toulouse, France, Jun. 2018. [Online]. Available: https://hal.inria.fr/hal-01871531

[5] G. Liboni, J. Deantoni, A. Portaluri, D. Quaglia, and R. De Simone, "Beyond Time-Triggered Co-simulation of Cyber-Physical Systems for Performance and Accuracy Improvements," in *10th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, Manchester, United Kingdom, Jan. 2018. [Online]. Available: https://hal.inria.fr/hal-01675396

[6] F. Cremona, M. Lohstroh, D. Broman, M. Di Natale, E. A. Lee, and S. Tripakis, "Step Revision in Hybrid Co-simulation with FMI," in *14th ACM-IEEE International Conference on Formal Methods and Models for System Design*. Kanpur, India: IEEE, Nov. 2016.

[7] F. Cremona, M. Lohstroh, S. Tripakis, C. Brooks, and E. A. Lee, "FIDE: An FMI integrated development environment," in *31st Annual ACM Symposium on Applied Computing*. Pisa, Italy: ACM New York, NY, USA, 2016, pp. 1759–1766.

[8] J.-P. Tavella, M. Caujolle, S. Vialle, C. Dad, C. Tan, G. Plessis, M. Schumann, A. Cuccuru, and S. Revol, "Toward an accurate and fast hybrid multi-simulation with the FMI-CS standard," in *21st IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Berlin, Germany: IEEE, Sep. 2016, pp. 1–5.

[9] S. Mustafiz, C. Gomes, H. Vangheluwe, and B. Barroca, "Modular design of hybrid languages by explicit modeling of semantic adaptation," in *2016 Symposium on Theory of Modeling and Simulation (TMS-DEVS)*, April 2016, pp. 1–8.

[10] B. Van Acker, J. Denil, H. Vangheluwe, and P. De Meulenaere, "Generation of an optimised master algorithm for fmi co-simulation," in *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, ser. DEVS '15. San Diego, CA, USA: Society for Computer Simulation International, 2015, pp. 205–212. [Online]. Available: http://dl.acm.org/citation.cfm?id=2872965.2872993

[11] S. Tripakis, D. Broman, and C. Sciences, "Bridging the Semantic Gap Between Heterogeneous Modeling Formalisms and FMI," Tech. Rep., 2014.

[12] D. Garlan and M. Shaw, "Introduction to software architecture," *Advanced Topics in Science and Technology in China*, no. January, pp. 1–33, 1994.

[13] N. Medvidovic and R. N. Taylor, "A framework for classifying and comparing architecture description languages," *ACM SIGSOFT Software Engineering Notes*, vol. 22, no. 6, pp. 60–76, 2004.

[14] G. A. Papadopoulos and F. Arbab, "Coordination Models and Languages," *Advances in Computers*, vol. 46, no. C, pp. 329–400, 1998.

[15] E. A. Lee and A. Sangiovanni-Vincentelli, "A framework for comparing models of computation," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 17, no. 12, pp. 1217–1229, 1998.

[16] C. Hardebolle and F. Boulanger, "Modhelx: A component-oriented approach to multi-formalism modeling," in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2007, pp. 247–258.

[17] M. E. Vara Larsen, J. Deantoni, B. Combemale, and F. Mallet, "A Behavioral Coordination Operator Language (BCOoL)," in *International Conference on Model Driven Engineering Languages and Systems (MODELS)*, T. Lethbridge, J. Cabot, and A. Egyed, Eds., no. 18. Ottawa, Canada: ACM, Sep. 2015, p. 462, to be published in the proceedings of the Models 2015 conference. [Online]. Available: https://hal.inria.fr/hal-01182773

[18] E. Bousse, T. Degueule, D. Vojtisek, T. Mayerhofer, J. Deantoni, and B. Combemale, "Execution framework of the gemoc studio (tool demo)," in *Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering*. ACM, 2016, pp. 84–89.

[19] J. S. Dahmann, "High level architecture for simulation," in *Proceedings First International Workshop on Distributed Interactive Simulation and Real Time Applications*. IEEE, 1997, pp. 9–14.

[20] J. Deantoni and C. Gomes, "Towards a ultimate formally verified master algorithm," *Short Term Scientific Report COST IC1404*, 2018.

[21] B. Combemale, J. Deantoni, M. E. Vara Larsen, F. Mallet, O. Barais, B. Baudry, and R. France, "Reifying Concurrency for Executable Metamodeling," in *SLE - 6th International Conference on Software Language Engineering*, ser. Lecture Notes in Computer Science, Erwig, Martin, Paige, R. F., V. Wyk, and Eric, Eds., vol. 8225. Indianapolis, IN, United States: Springer, Oct. 2013, pp. 365–384. [Online]. Available: https://hal.inria.fr/hal-00850770

[22] OMG, "Uml profile for marte (modeling and analysis of real time embedded systems)," *Object Management Group*, vol. v1.2, Dec. 2018.

[23] C. André, "Syntax and semantics of the clock constraint specification language," INRIA, Tech. Rep. 6925, 2009.

[24] P. Le Guernic, J.-P. Talpin, and J.-C. Le Lann, "Polychrony for system design," *Journal of Circuits, Systems and Computers*, vol. 12, no. 03, pp. 261–303, 2003.