Towards Reuse of Synchronization Algorithms in Co-simulation Frameworks*

Casper Thule[1], Maurizio Palmieri[2][3], Cláudio Gomes[4], Kenneth Lausdahl[5], Hugo Daniel Macedo[1], Nick Battle[6], and Peter Gorm Larsen[1]

[1] DIGIT, Department of Engineering, Aarhus University, Finlandsgade 22, 8200 Aarhus N, Denmark, {casper.thule, hdm, pgl}@eng.au.dk, http://digit.au.dk
[2] University of Florence,
[3] Pisa University, maurizio.palmieri@ing.unipi.it
[4] University of Antwerpen, claudio.gomes@uantwerp.be
[5] Mjølner Informatics A/S, kgl@mjolner.dk
[6] Independent, nick.battle@acm.org

Abstract. An immediate industry challenge is to fashion a co-simulation that replicates real systems behaviour with high fidelity. To achieve this goal, developers rely on frameworks to enhance the creation and analysis of the co-simulation. The problem is that new co-simulation frameworks require extensive development, most of which in non-essential functionalities, before they can be used in practice. Additionally, existing co-simulations demand a thorough understanding before they can be extended.
Our vision is a modular co-simulation framework architecture, that is easily extensible by researchers, and can integrate existing and legacy co-simulation approaches. The architecture we propose permits extension at three levels, each providing different degrees of flexibility. The most flexible integration level involves the specification of a Domain Specific Language (DSL) for Master Algorithm (MA) and this paper sketches such DSL, and discusses how it is expressive enough to describe well-known MAs.

Keywords: co-simulation · Functional Mock-up Interface · co-simulation framework

1 Introduction

Co-simulation frameworks require extensive development before they can become usable in practice. A co-simulation following the Functional Mock-up Interface (FMI) Standard is a collaborative simulation carried out by combining multiple simulators called Functional Mock-up Units (FMUs), each representing a constituent of a system [7]. The algorithm describing how the coupling of

* This work has been supported by the Poul Due Jensen Foundation. C. G. is a FWO Research Fellow, at the University of Antwerp, supported by the Research Foundation - Flanders (File Number 1S06316N).

FMUs is carried out is referred to as the MA, and the FMUs are denoted as slaves. A scenario is the configuration of which FMUs to use, how the FMUs are coupled, and which MA to use[7].

A co-simulation framework provides the foundations that implement the elements above, allowing users to run co-simulations, and execute other simulation activities such as optimization, sensitivity analysis, etc. In essence, such a framework is what makes co-simulation an integral part of a development process. To be usable, therefore, a co-simulation framework needs to seamlessly integrate with existing design processes. For that, its users need not worry about:

1. how to establish communication with the FMUs; or
2. how to configure the MA and FMUs, to achieve reliable co-simulation results.

Fortunately, co-simulation standards such as the FMI have largely relieved practitioners from having to worry about the former. As for the latter, however, recent surveys [7, 13, 16, 21, 20] indicate that the configuration of an MA is still an open challenge. For instance, the configuration of scenarios is one of the immediate industry challenges [20], and there is evidence that reliable co-simulation results might not be, in general, attainable without a custom combination of existing co-simulation results [17, 9].

To research and develop novel co-simulation approaches, it is necessary to equip researchers with proper foundations to conduct their research on. This means relieving them from the need for extensive development efforts that are not directly related to co-simulation configuration and execution. In fact, the development team of the INTO-CPS application [14] has reported that the development of the features surrounding the execution of a MA (e.g., simulator loading, Graphical User Interface (GUI), setup and deployment, etc) far exceed the effort of coding such MA. The development took place over three years, from 2015 to 2017, and the approximate time percentage spent on common functionality is shown in Fig. 1.

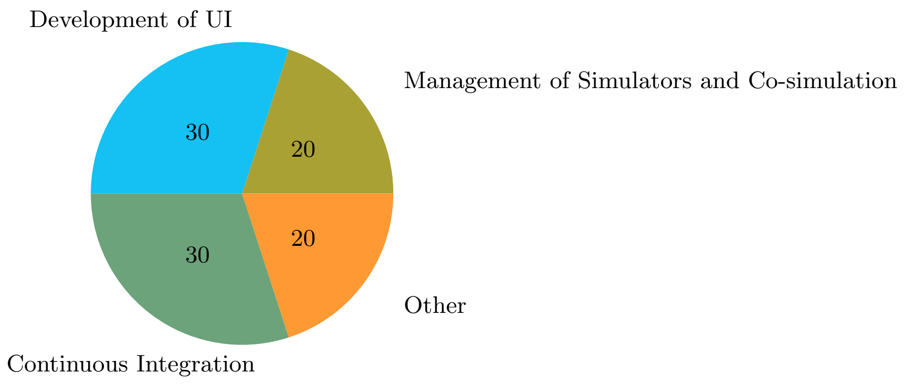

Fig. 1. Time(%) spent on developing common functionality.

[7] We adopt the terminology in [10].

Researchers looking to have an impact with their novel co-simulation approaches may never achieve it if they cannot afford such development effort. At the same time, existing co-simulation frameworks cannot keep up with the new developments in MAs, making the adoption of novel techniques slow.

Therefore, there is a need to modularize the architecture of a co-simulation framework, in order to maximise reuse, and combination, of mature and industry proven features common to every MA, in the deployment of novel MAs.

Contribution. In this paper, we propose an architecture to promote easy integration of novel MAs into a co-simulation framework. This architecture enables researchers to contribute with custom MAs with three different levels of integration, each providing increasing levels of flexibility. The most flexible integration level involves the specification of a DSL for MA and this paper proposes a preliminary analysis of requirements for such a DSL.

2 Problem Statement

This section exposes the problem that we are trying to solve. We assume that the reader is familiar with co-simulation (see, e.g., [10, 12] for an introduction and tutorial), and the FMI standard (see, e.g., [3] for an introduction, and [5] for the specification). We will adopt the following definitions:

FMU Runtime denotes the set of libraries that allows one to load, instantiate, and communicate with SUs. Examples are: INTO-CPS FMI library[8] (Java), the FMPy library[9] (Python), PyFMI[10] (Python), or the FMI Library[11] (C).

MA denotes the procedure that coordinates the time synchronization between FMUs. It relies on the FMU Application Programming Interface (API) to communicate with the FMUs. Examples are the Jacobi or Gauss Seidel algorithms.

GUI, Command Line Interface (CLI), API denote the interfaces that enable a users to describe the co-simulation scenario, to configure the MA, and to run co-simulations. Examples include the INTO-CPS Application[12] [19, 2].

Simulation Activities denotes any activity that is part of a development process and relies on the GUI/CLI/API to be completed. For example, optimization/Design Space Exploration (DSE), sensitivity analysis, or X-in-the-loop co-simulation.

Figure 2 summarizes the layered relationship of these concepts, and distinguishes a co-simulation framework from co-simulation application.

Traditionally, co-simulation has been applied to mostly pairs of simulators, with custom built MAs [11]. This worked well because the people who built

[8] https://github.com/INTO-CPS-Association/org.intocps.maestro.fmi

[9] https://github.com/CATIA-Systems/FMPy

[10] https://jmodelica.org/pyfmi/

[11] https://jmodelica.org/fmil/FMILibrary-2.0.3-htmldoc/index.html

[12] https://github.com/INTO-CPS-Association/into-cps-application

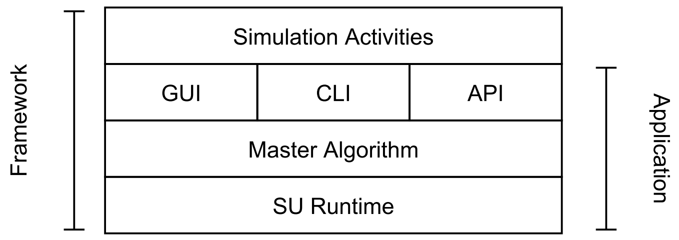

Fig. 2. Main definitions used.

those MAs were, or worked closely with, domain experts. However, this approach does not scale in the number of simulators involved, and the FMI standard was developed to address this need. As a result, many practitioners expect to use FMI co-simulation without having co-simulation expertise, and the black box nature of this standard (where the models being simulated, and solvers used, are kept hidden) does not make it easier to understand what are the simulators that are being coupled.

In fact, there is evidence that version 2.0 of the standard is insufficient to ensure that a co-simulation can be configured correctly, and that more information about the FMUs is required[13]: a clear indication that research in co-simulation will continue, and that researchers will have to code new co-simulation frameworks. We want to minimize the effort required for these researchers to produce usable frameworks.

Just as with simulation, it is instructive to run multiple co-simulations with different MAs, to measure the degree of sensitivity of the results with respect to the MA and simulator configuration. Therefore, we also want to be able to produce a unified front-end to users who want to run such exploratory co-simulations.

Figure 3 shows the space of simulator information that can be used to configure co-simulations, and the space of capabilities currently covered by the FMI standard (version 2.0). Our goal is that co-simulations taking advantage of these extra capabilities, even the ones not covered by the standard, can be developed. Moreover, in the long term, we aim at improving the co-simulation support for the following simulation activities, each imposing specific requirements to co-simulation frameworks:

Optimization/DSE: Co-simulations are run as part of an optimization loop. This includes decision support systems, used, for example, in a digital twin setting. Some of the specific requirements include: ability to define co-simulation

[13] See [9, 17, 6] for example co-simulations that cannot be configured correctly without information that is not covered in the standard.

stop conditions, ability to compute sensitivity, high performance, fully automated configuration, faster than real-time computation.

Certification: Co-simulation results are used as part of certification purposes. Requirements include fully transparent, and formally certified, synchronization algorithms.

X-in-the-loop: Co-simulations include simulators that are constrained to progress in sync with the wall-clock time, because they represent human operators or physical subsystems.

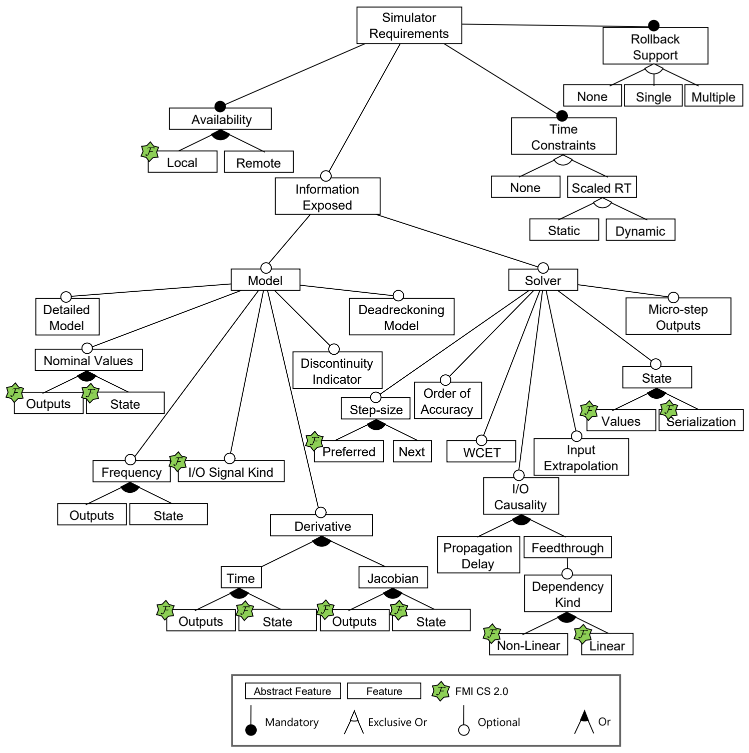

Fig. 3. Simulator capabilities. The FMI standard capabilities represent only a subset, and many of these are optional. Adapted from [11].

3 Envisioned Architecture

Our goal is to conceive an architecture which can be evolved to accommodate a wide variety of co-simulation improvements. Figure 4 summarizes the proposed architecture, and will be used as reference in the rationale and explanation below.

3.1 Legacy Integration

The legacy integration, to the left of Fig. 4, involves the improvement of the existing interface between the INTO-CPS application, so that existing MAs can be integrated. Under this approach, a legacy MA, along with its own simulator runtime libraries, uses the Master API to communicate with the INTO-CPS Application.

Legacy integration is already partially supported by the MA API that has been developed within the INTO-CPS project [22]. The Master API is detailed in [18]. Any new MA needs to implement one of those. The following are some of the operations.

Status which allows the Application to query the status of the MA.
Initialize which allows new co-simulation sessions to be created. A JSON payload details the co-simulation scenario and other configuration parameters.
Simulate which instructs the MA to start the co-simulation, provided the experiment parameters.
Result which queries the MA for the simulation results.
Destroy which instructs the MA to clear the resources of a session.
Reset which instructs the MA to reset a session.

3.2 MA Integration

The MA integration, in the middle of Fig. 4, unlike the legacy integration, requires only that the new MA implements the Master API, and uses the provided Runtime API for the management of FMUs.

Such API allows the new MA to easily instantiate FMUs, manage their lifecycle, and inspect their information. For example, the new MA will not need to parse the FMU description in order to access the available variables.

3.3 Approach Integration

Finally, the most ambitious of the integration schemes is aimed at researchers who want to quickly develop and test new co-simulation approaches. We envision the development of an extensible DSL, denoted by Master Specification Language, that aims at expressing synchronization algorithms. The rationale is that there are many operations which are common to all MAs, e.g., the act of rolling back to a previous time point, or of retrying the co-simulation step. This language will separate the planning of the co-simulation approach, from the execution of such plan, freeing researchers from having to specify how the operations of a synchronization scheme are executed, focusing only on developing algorithms to describe what those operations are. Furthermore, the developed APIs will allow for analysis/optimization plugins to be integrated as well. For example, one can have an analysis that enforces a specific version of the FMI standard.

The main difference with respect to the previous mode of integration is that a new MA, instead of invoking the runtime API to run the co-simulation, will produce a sequence of instructions detailing how the simulator synchronizes. This sequence of instructions, denoted the synchronization protocol, will be produced through a series of transformations that are applied to the co-simulation scenario given.

More details are given in Section 4.

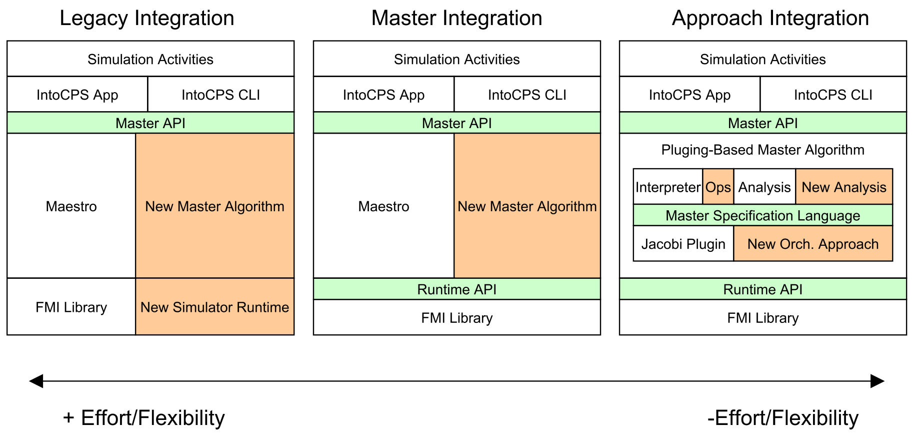

Fig. 4. The three planned integration approaches. Each approach is represented as a layered architecture, and an example of given of a newly integrated component (in orange). In green we represent the key interfaces that enable the integration. Components below the API implement it, while the ones above, use it.

4 MA Specification Language

In this section, we provide a preliminary specification of the DSL introduced in the previous section. The language is comprised of three main parts: synchronization protocols; scenario and adaptations; and transformations (which describe how a co-simulation scenario is transformed into a MA). Each part is now described. We end this section with a discussion on the extensible part of the DSL.

4.1 Synchronization Protocols

We start by giving examples of how known co-simulation algorithms can be implemented in this framework, and then we generalize to the specification of the language.

Example 1 (Running Example). We will use, as running example, the co-simulation scenario illustrated in Fig. 5.

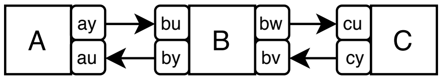

Fig. 5. Running example. The rectangles define FMUs and the round rectangles denote inputs/outputs.

A generic co-simulation algorithm has a predictable structure, shown in Fig. 6, and variations in the implementation of each of the stages shown will yield different MAs. Our DSL allows one to describe the structure of the Step and Initialize stages of Fig. 6. We focus on the Step stage, as it is the richest in terms of variability, and the Initialize activity can be seen as a special case of the Step stage.

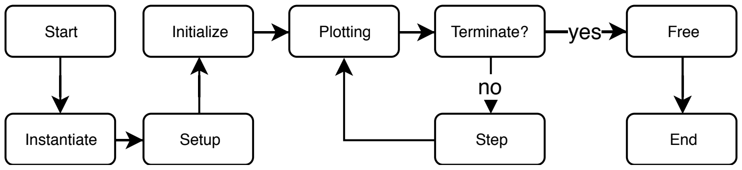

Fig. 6. Generic MA structure. Each round rectangle represents a stage in the execution of the MA. For instance, the plotting stage will query the outputs of the FMUs and record them in a CSV file.

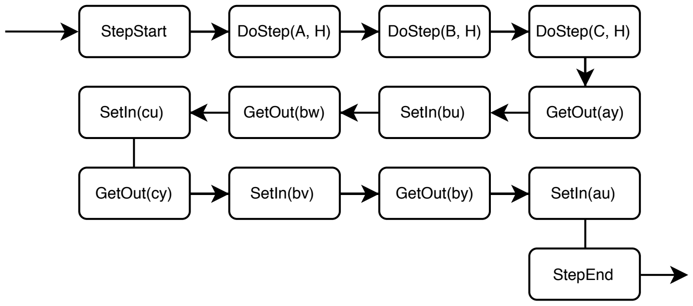

Fig. 7. Jacobi Step specification, applied to Example 1.

The simpler MAs are those with fixed step sizes. Well known examples are the Jacobi and the Gauss-Seidel. The Jacobi MA, applied to Example 1, is shown in Fig. 7. Each operation is translated to the corresponding FMI function call, with the DSL implementation filling in the missing arguments: the simulated time, used in the FMI stepping operation, is automatically computed; the values computed by the `GetOut` operations are stored in variables that are then used to compute the correct value for the `SetIn` operations. The management of values is implemented by translating the `GetOut` and `SetIn` operations to their refined counterparts, as is shown in Fig. 8.

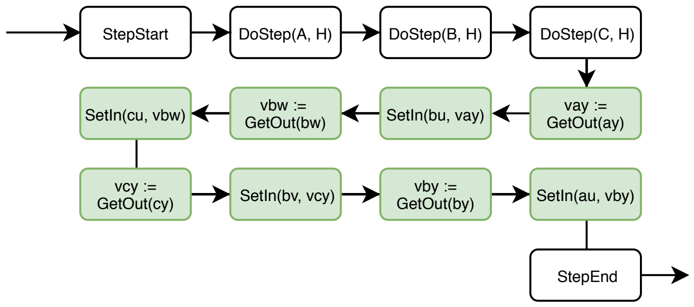

Fig. 8. Explicit memory refinement of Fig. 7.

The MA in Fig. 8 can be enhanced in a number of ways:

– Automate the choice of step size to use at each step to control the error, rolling back with the error is deemed too large.
– Dealing with an FMU that refuses a given step.
– Use apply fixed point iteration of algebraic conditions and/or the co-simulation step.

The insight of our contribution is that the above items can be orthogonality combined by refinements of the Step specification (e.g., the one shown in Fig. 7). An example for each of the above enhancements is provided in the following, by showing the resulting refinement.

Figure 9 shows an example of a pessimistic adaptive set size control scheme. The implementation of the `UpdateStep` operation can be defined in a plugin (see Section 4.4). The `Transaction`, `Commit`, and `Rollback` operations are implemented in the DSL.

Figure 10 shows an example that handles step size rejections, and Fig. 11 an example of fixed point iteration with convergence testing on one signal (multiple signals can be supported, but its implementation is a straightforward extension of Fig. 11).

4.2 Scenarios and Adaptations

The synchronization protocol sub language allows one to describe a wide range of co-simulation algorithms. However, it is not expressive enough to describe the following MAs: multi-rate; signal corrective; waveform relaxation; MAs that spawn new co-simulations (e.g., for each step of an FMU, an entire co-simulation is run

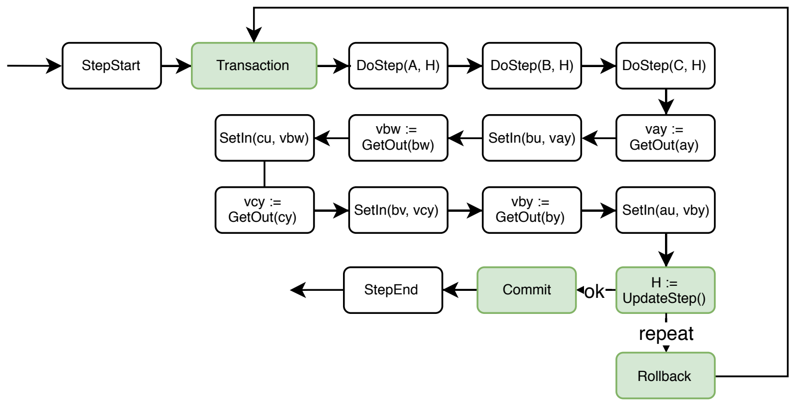

Fig. 9. Pessimistic adaptive step size MA.

to calculate some result). We now describe how the use of semantic adaptations which, when carefully combined with the synchronization protocol sub-language, can be used to implement the above algorithms.

Multi-rate. A semantic adaptation is a transformation that rewrites a co-simulation scenario by either changing a group of, or adding more, FMUs (see [8] for examples of semantic adaptations). For instance, a multi-rate adaptation consists of grouping a given set of FMUs into one hierarchical FMU, where the implementation of the latter ensures that the group of FMUs communicate at a higher rate than the rest of the co-simulation, as illustrated in Fig. 12. The multi-rate adaptation transformation can be applied to a scenario prior to passing the scenario to the synchronization protocol generation, explained in Section 4.1.

Signal-Corrective. Regarding signal corrective adaptations, we describe here the energy preserving adaptation, first described in [4]. The adaptation applies to a pair of connections that form a power bond (see [1, Chapter 9] for an introduction). Whenever a value is propagated in those connections, it gets corrected to account for approximation errors made in the receiving FMU. Hence, the adaptation replaces the ports connected by the power bond to apply that correction whenever an input is set. Alternatively, a new FMU that performs the correction can be inserted in place of the power bond connections. These are illustrated in Fig. 13.

The application of the energy preserving adaptation depends on the implementation of the synchronization protocol. As such the resulting scenario (after applying the adaptation) cannot be given to any synchronization protocol generator.

Waveform Relaxation. A waveform relaxation algorithm is an iterative co-simulation synchronization protocol that, instead of applying a fixed point iteration of point

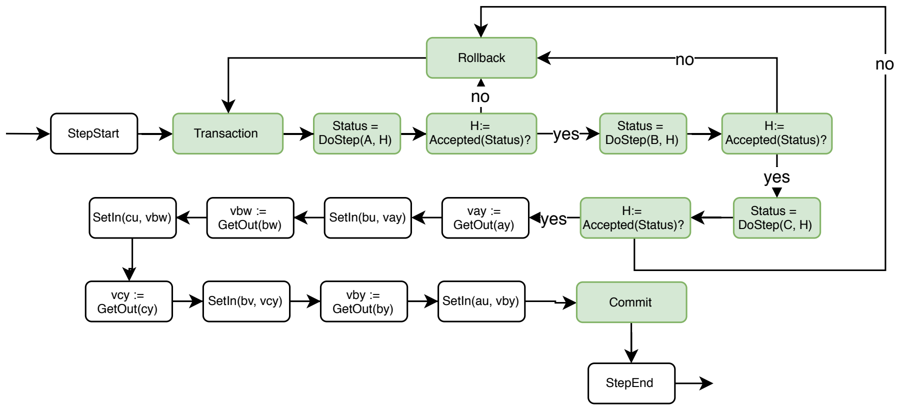

Fig. 10. Step specification that handles step size rejection.

values (as the iterative Jacobi described in Fig. 11 does), it applies a fixed point iteration on functions. There are multiple ways to check for equality of two functions. One of those ways is to perform a point-wise comparison, and return the maximum of such comparisons. Two functions are then considered equal if that value is within some given threshold.

The waveform relaxation is a kind of multi-rate adaptation that is combined with an iterative synchronization protocol (see [15] for an introduction). First, as illustrated in Fig. 14, each FMU is grouped into a multi-rate FMU, along with new FMUs that represent proxies of its environment. The multi-rate FMU will run a complete co-simulation each time it is invoked. When running this co-simulation, the Proxy FMUs will record the outputs of the non proxy FMU, and construct a function with these. The proxy FMUs also "replays" the input function they have in their storage. When this co-simulation is over, it means that the multi-rate FMU has completed a step. At that moment, the multi-rate FMUs will exchange input and output values. These values are complete functions, which are tested for convergence (recall Fig. 11). If they have converged, the co-simulation is over. Otherwise, the process is repeated, with the proxies having exchanged the recorded functions.

Sub-co-simulations. Finally, co-simulation scenarios that have FMUs that may spawn a new co-simulation are constructed with a Hierarchical Cosim FMU. Figure 15 illustrates an example of this. At each DoStep of the hierarchical cosim, a new co-simulation is run, with the parameters defined by the input value bu.

4.3 Transformations

The application of semantic adaptations, as introduced in Section 4.2, require careful coordination with the application of synchronization protocols, intro-

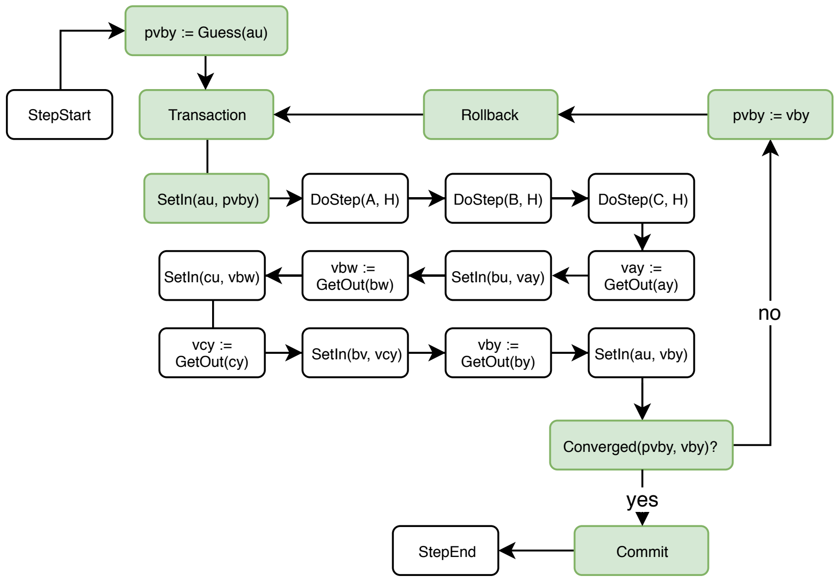

Fig. 11. Iterative refinement of the Jacobi step specification, with convergence test on one signal.

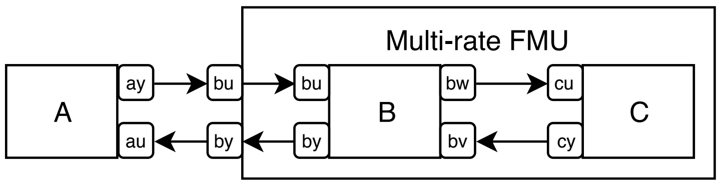

Fig. 12. Example multi-rate adaptation created by rewriting the scenario introduced in Fig. 5. FMUs B and C communicate at a higher rate.

duced in Section 4.1. Attempting to automatically derive the rules of such compositions is a tremendous challenge, and subject of future work. Instead, we support a library of transformations that can be composed manually to generate a MA. Such transformations operate on models described in our DSL.

A model contains a scenario, adaptations, and a MA, as is illustrated in Fig. 16. A particular co-simulation approach is then a sequence of transformations applied to a model that yield an executable co-simulation algorithm.

A composition of transformations describes how the model is refined until it can be executed. For example, a multi-rate composition would first apply the multi-rate adaptation to rewrite the scenario in Fig. 16 to the scenario in Fig. 12, and then would apply the Jacobi algorithm transformation to refine the step operation in Fig. 16 to Fig. 7. Different sequences of transformations will yield potentially different MAs.

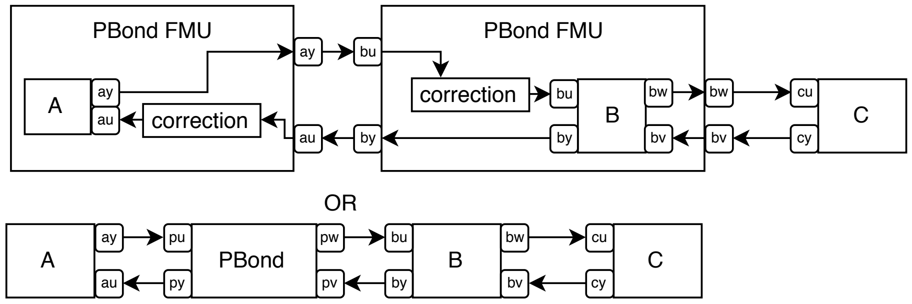

Fig. 13. Illustration of two implementations of the energy correction adaptation.

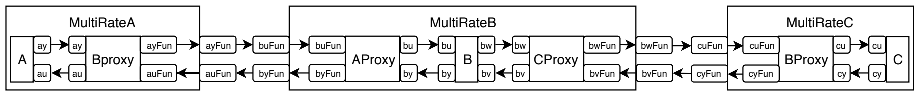

Fig. 14. Illustration of waveform relaxation adaptation, applied to Example 1.

4.4 Extensibility

There are several ways to extend the DSL:

Protocol Operation New synchronization operations can be declared. Our interpreter will then load the declared plugins and execute them. For example, the `UpdateStep` operation, in Fig. 10, could be implemented in a plugin.

Virtual FMUs Virtual FMUs, implemented in Scala or Java, can be declared. These will be loaded and executed by the interpreter as any other FMUs, with the difference that they do not need to be represented as FMUs in the FMI standard.

Transformation Rules New transformation rules that manipulate the co-simulation scenario, and/or the synchronization protocol, can be declared.

Transformation Compositions New transformation compositions can be declared.

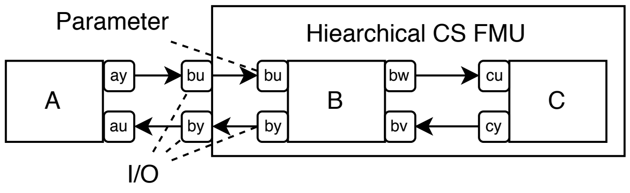

Fig. 15. Illustrations of the use of a hierarchical FMU to run a co-simulation between B and C, for each step in the co-simulation of A.

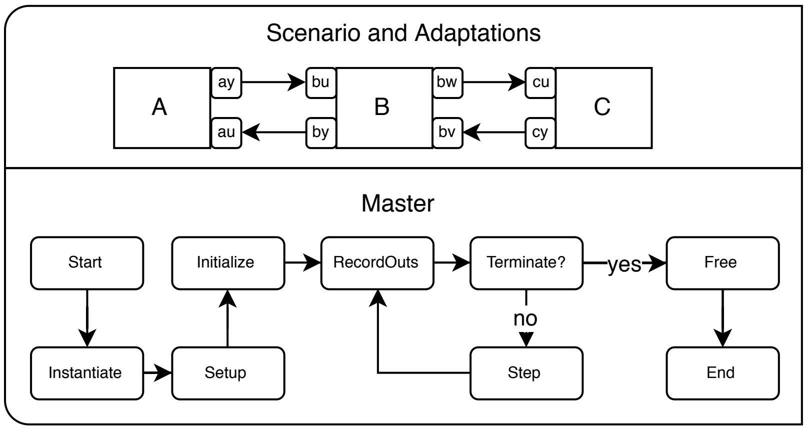

Fig. 16. Example model.

We envision these extensions to be done in the form of plugins, without the need to recompile the tool. The rationale for this requirements is that researchers experimenting with novel master algorithms need not be proficient with building and deploying the DSL.

5 Prospect and Future Work

We have sketched an architecture to support multiple levels of integration, in order to maximize the reuse of existing co-simulation algorithms, and facilitate the development of new ones. Our main contribution is a DSL that allows novel co-simulation algorithms to be developed, by refining a given scenario and a generic master algorithm. A preliminary prototype has been built defining the semantics of simple synchronization protocols. Ongoing work is formalizing the language. A general goal is that the language and extension interfaces shall consistently be in a stable and usable state once the initial plugins and native functionality to conduct a co-simulation has been realized.

An interesting research opportunity is to devise analysis that ensure the validity of arbitrary transformation compositions. Such analysis would be a first step towards deriving rules for automated enhancement of master algorithms provided by researchers (for example, the addition of step size rejection handlers). The final goal that we foresee is a free marketplace where plugins are selected and composed to produce MAs addressing specific needs.

We expect to validate the DSL by developing master algorithms that make use of vendor specific information in FMUs to achieve better results, and enable co-simulation within the digital twin context. This will require the use of the multi-rate and hierarchical co-simulation adaptations, plus custom synchronization operations.

References

1. van Amerongen, J.: Dynamical Systems for Creative Technology. Controllab Products B.V., http://doc.utwente.nl/75219/
2. Bandur, V., Larsen, P.G., Lausdahl, K., Thule, C., Terkelsen, A.F., Gamble, C., Pop, A., Brosse, E., Brauer, J., Lapschies, F., Groothuis, M., Kleijn, C., Couto, L.D.: INTO-CPS Tool Chain User Manual. Tech. rep., INTO-CPS Deliverable, D4.3a (December 2017)
3. Bastian, J., Clauß, C., Wolf, S., Schneider, P.: Master for Co-Simulation Using FMI. In: 8th International Modelica Conference. pp. 115–120. Linköping University Electronic Press, Linköpings universitet. https://doi.org/10.3384/ecp11063115
4. Benedikt, M., Watzenig, D., Zehetner, J., Hofer, A.: NEPCE-A Nearly Energy Preserving Coupling Element for Weak-coupled Problems and Co-simulation. In: IV International Conference on Computational Methods for Coupled Problems in Science and Engineering, Coupled Problems. pp. 1–12
5. Blochwitz, T., Otter, M., Åkesson, J., Arnold, M., Clauss, C., Elmqvist, H., Friedrich, M., Junghanns, A., Mauss, J., Neumerkel, D., Olsson, H., Viel, A.: Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In: Proceedings of the 9th International Modelica Conference. pp. 173–184. The Modelica Association (2012). https://doi.org/10.3384/ecp12076173, key=blo+12mc project=LCCC-modeling
6. Cremona, F., Lohstroh, M., Broman, D., Lee, E.A., Masin, M., Tripakis, S.: Hybrid co-simulation: It's about time **10270**. https://doi.org/10.1007/s10270-017-0633-6
7. Gomes, C., Thule, C., Broman, D., Larsen, P.G., Vangheluwe, H.: Co-simulation: a Survey. ACM Comput. Surv. **51**(3), 49:1–49:33 (May 2018)
8. Gomes, C., Meyers, B., Denil, J., Thule, C., Lausdahl, K., Vangheluwe, H., De Meulenaere, P.: Semantic Adaptation for FMI Co-simulation with Hierarchical Simulators **95**(3), 1–29. https://doi.org/10.1177/0037549718759775
9. Gomes, C., Oakes, B.J., Moradi, M., Gamiz, A.T., Mendo, J.C., Dutre, S., Denil, J., Vangheluwe, H.: HintCO - Hint-Based Configuration of Co-Simulations. In: International Conference on Simulation and Modeling Methodologies, Technologies and Applications. p. accepted
10. Gomes, C., Thule, C., Broman, D., Larsen, P.G., Vangheluwe, H.: Co-simulation: A Survey **51**(3), Article 49. https://doi.org/10.1145/3179993
11. Gomes, C., Thule, C., Broman, D., Larsen, P.G., Vangheluwe, H.: Co-simulation: State of the art, http://arxiv.org/abs/1702.00686
12. Gomes, C., Thule, C., Larsen, P.G., Denil, J., Vangheluwe, H.: Co-simulation of Continuous Systems: A Tutorial, http://arxiv.org/abs/1809.08463
13. Hafner, I., Popper, N.: On the terminology and structuring of co-simulation methods. In: Proceedings of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools. pp. 67–76. ACM Press. https://doi.org/10.1145/3158191.3158203
14. Larsen, P.G., Fitzgerald, J., Woodcock, J., Fritzson, P., Brauer, J., Kleijn, C., Lecomte, T., Pfeil, M., Green, O., Basagiannis, S., Sadovykh, A.: Integrated tool chain for model-based design of Cyber-Physical Systems: The INTO-CPS project. In: 2016 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data). IEEE, Vienna, Austria (April 2016), http://ieeexplore.ieee.org/document/7496424/
15. Li, L., Seymour, R.M., Baigent, S.: Integrating biosystem models using waveform relaxation **2008**, 308

16. Palensky, P., Van Der Meer, A.A., Lopez, C.D., Joseph, A., Pan, K.: Cosimulation of Intelligent Power Systems: Fundamentals, Software Architecture, Numerics, and Coupling **11**(1), 34–50. https://doi.org/10.1109/MIE.2016.2639825
17. Pedersen, N., Lausdahl, K., Sanchez, E.V., Thule, C., Larsen, P.G., Madsen, J.: Distributed Co-simulation of Embedded Control Software Using INTO-CPS, pp. 33–54. Springer International Publishing, Cham (2019)
18. Pop, A., Bandur, V., Lausdahl, K., Thule, C., Groothuis, M., Bokhove, T.: Final Integration of Simulators in the INTO-CPS Platform. Tech. rep., INTO-CPS Deliverable, D4.3b (December 2017)
19. Rasmussen, M.B., Thule, C., Macedo, H.D., Larsen, P.G.: Moving the INTO-CPS Application to the Cloud. In: Submitted for publication
20. Schweiger, G., Gomes, C., Engel, G., Hafner, I., Schoeggl, J., Posch, A., Nouidui, T.: Functional Mock-up Interface: An empirical survey identifies research challenges and current barriers. In: The American Modelica Conference. pp. 138–146. Linköping University Electronic Press, Linköpings universitet, Cambridge, MA, USA (2018). https://doi.org/10.3384/ecp18154138
21. Schweiger, G., Gomes, C., Engel, G., Hafner, I., Schoeggl, J.P., Posch, A., Nouidui, T.: An empirical survey on co-simulation: Promising standards, challenges and research needs **95**, 148–163. https://doi.org/10.1016/j.simpat.2019.05.001
22. Thule, C., Lausdahl, K., Gomes, C., Meisl, G., Larsen, P.G.: Maestro: The INTO-CPS Co-simulation Framework **92**, 45–61. https://doi.org/10.1016/j.simpat.2018.12.005