UNIVERSITEIT ANTWERPEN

Property Preservation in Co-simulation

Eigenschapsbewaring in co-simulatie

Auteur:
Cláudio GOMES

Promotor:
Prof. Dr. Hans VANGHELUWE
Co-Promotor:
Prof. Dr. Paul DE MEULENAERE

Proefschrift ingediend tot het behalen van de graad van
Doctor in de Wetenschappen: Informatica

Contents

List of Figures

List of Tables

Acknowledgements

For the past five years I've crossed paths with many people whose contribution to this work cannot be understated. What follows is an attempt at expressing my gratitude.

I am forever indebted to my supervisor, Hans Vangheluwe, for always taking the extra time for an interesting discussion, and for all the pearls of wisdom dropped at any time (e.g., while driving in a busy highway, or while trying to fall asleep after a long day). Most importantly, thank you for the contagious enthusiasm, for all the hours you put in understanding the topic and its ramifications for future research, and for the opportunities you created for me.

My co-supervisor, Paul De Meulenaere, also played a very important role in this work. Thank you Paul for always making time for a meeting, for your feedback, and for your advice on how to bring this work to industry.

I am also thankful for the many meetings I had with Peter Gorm Larsen. You have changed the course of my work through your advice and the opportunities you created. Thank you for, even in the busiest of times, taking the time to improve our manuscripts. I could not have survived in Denmark if it weren't for the hospitality of you and your wife (Margit Larsen), and the wonderful meals.

Financially, this thesis would not have been possible without the support of the Research Foundation - Flanders (FWO), and I would not have been able to produce the cover without the pictures that are available in the public domain (websites *freepik* and *unsplash*).

Socially, I cannot express how important my friends have been in this work:

- Thank you Simon, Ali, Yentl, Diana, and Bentley, for the lunch discussions for all the papers proposing to solve pertinent world's problems, that went unwritten, because the lunch break is too short;
- Thank you Joachim, Ken, Istvan, and Bart, for all the great brainstorming sessions and ideas that later guided much of our work, and sharing your wisdom with me;
- Thank you Benoît (and Lea), and Raphaël, for believing in the relevance of our work, even when you had other research to do, for your incredible patience in helping me see the beauty of so many mathematical results, for attempting to teach me how to write for mathematical audiences, and for making sure I had a great time when visiting Louvain-la-Neuve.
- Thank you Casper and Kenneth, for teaching me so much about large scale software development, and all the ideas that made into our work together.

- Thank you Alex, Mendo, Satya, and Stefan, for helping me understand the needs of industry on co-simulation.
- And a big thank you to all the friends I visited (Levi, André, Daniela, Regina, Citra), and visited me (Cláudia, Silvia, João, Vasco, Carmelita, Youyou, Rúben, Valter, Julien), who brought me so many treats from all over the world.

I am grateful to have a family that supported me in my decision to pursue a PhD, even though this meant being far from home. Maybe this is their way of getting rid of me. Nevertheless, I cannot imagine how difficult this is for parents and grandparents, and I just hope the results I achieve make up for their sacrifice.

Finally, I thank you Caroline Manik, for all the dry-runs you brace through, the weekends we sit in, the serenity with which you listen to my complaints, your wise advice, and for patiently showing me the beauty of economics. Thank you for your love and the home you created.

Cláudio Gomes
1 March 2019

Abstract

Modeling and Simulation (M&S) techniques are today extensively used both in industry and science, to develop and understand complex systems. As development processes morph to respond to economic pressures, they impose new demands on these techniques: • frequent full system evaluation is required to prevent late integration problems among specialized teams who worked in parallel on different, but interconnected, parts of the system; and • seamless integration of externally supplied component models into the M&S workflow of Original Equipment Manufacturers is needed to enable high fidelity simulations.

Traditional M&S techniques, where a single model of the whole system is built and simulated, are insufficient to address these demands, because: • teams use mature M&S tools, each tailored to a particular domain, and not capable of exporting models that are compatible with any other M&S tool; and • external suppliers are not willing to share high fidelity models without expensive contracts protecting their Intellectual Property (IP).

Co-simulation is a way to tackle these challenges. It consists of the theory and techniques to enable global simulation of a coupled system via the composition of simulators. Each simulator is broadly defined as a *black box* capable of exhibiting behavior, consuming inputs and producing outputs.

This nature is also what aggravates the fundamental challenge in co-simulation: deciding whether the results can be trusted.

This thesis is comprised of two parts. The first part explores the challenge, and tries to understand what makes co-simulation different than traditional simulation techniques. One of the conclusions of this part is the need to ensure that co-simulations preserve properties of the system being developed (e.g., stability, smoothness, etc…). The second part represents a collection of work, each targeting an aspect of property preservation, including giving the users of co-simulation, the ability to control the implementation of participating simulators, and then providing a framework to help them do so correctly.

The reported results provide a deeper understanding of the fundamental challenges that need to be addressed before co-simulation can become a seamless technique in the development of complex systems, including: 1. how to configure adaptive co-simulation algorithms that preserve stability; 2. how to configure state event location for co-simulation of hybrid systems; 3. a tool that allows the configuration of simulators participating in a co-simulation; and 4. a framework that guides the configuration of the co-simulation, so as to preserve domain specific description of system properties.

Nederlandstalige Samenvatting

Technieken voor het Modelleren en Simuleren (M&S) van complexe systemen om ze te ontwerpen en verstaan worden meer en meer gebruikt in de industrie en wetenschap. Naarmate ontwikkelprocessen evolueren om een antwoord te bieden aan economische druk, worden er nieuwe vereisten gesteld aan deze technieken: • het frequent evalueren van het volledige systeem is nodig om problemen bij late integratie te voorkomen bij gespecialiseerde teams die in parallel werken aan verschillende, maar geconnecteerde delen van het systeem; en • er is nood aan de naadloze integratie van extern geleverde modellen van componenten in de workflow van *Original Equipment Manufacturers* (OEMs) om simulaties van hoge betrouwbaarheid te bekomen.

Traditionele M&S technieken, waar één model van het systeem wordt gebouwd en gesimuleerd, volstaan niet om aan deze vereisten te voldoen, want: • hooggespecialiseerde teams gebruiken tools die specifiek zijn voor een bepaald domein, waarvan de modellen niet geëxporteerd kunnen worden naar andere M&S tools; en • externe leveranciers willen hun modellen met hoge betrouwbaarheid niet delen zonder dure contracten af te sluiten die hun *Intellectual Property* (IP) beschermen.

Co-simulatie biedt een oplossing om deze moeilijkheden te overkomen. Het bestaat uit een theorie en technieken die een globale simulatie toelaten van een gekoppeld systeem door het koppelen van simulators. Elke simulator is grofweg gedefinieerd als een *zwarte doos* die gedrag kan hebben, waarbij input geconsumeerd wordt en output geproduceerd wordt. Deze natuur van simulatoren bemoeilijkt de fundamentele uitdagingen in co-simulatie: beslissen of de resultaten betrouwbaar zijn.

Deze thesis bestaat uit twee delen. Het eerste deel verkent de uitdagingen, en probeert de verstaan hoe co-simulatie verschilt van traditionele simulatietechnieken. Eén van de conclusies van dit deel is de noodzaak dat co-simulaties de eigenschappen van het systeem dat wordt ontwikkeld bewaart (e.g., stabiliteit, gladheid, etc. . .). Het tweede deel representeert een collectie van onderzoeksonderwerpen, die elk een aspect van eigenschapsbewaring behandelen, inclusief het geven van de mogelijkheid aan de gebruiker van de co-simulatie om de implementatie van de betrokken simulators aan te passen, en het levert een framework aan om dit correct te doen.

De gerapporteerde resultaten bieden een dieper inzicht in de fundamentele uitdagingen die moeten geadresseerd worden voordat co-simulatie naadloos kan gebruikt worden voor het

ontwikkelen van complexe systemen, inclusief: 1. hoe co-simulaties kunnen geconfigureerd worden om stabiliteit te behouden; 2. hoe het detecteren van toestandsveranderingsevenementen kan geconfigureerd worden voor hybride systemen; 3. een tool dat toelaat om simulatoren die deelnemen in een co-simulatie te configureren; en 4. een raamwerk dat de configuratie van de co-simulatie gidst, zodat de domeinspecifieke beschrijving van systeemeigenschappen behouden blijft.

Publications

The following peer-reviewed publications are partially included in this thesis:

- GOMES, CLÁUDIO. "Foundations for Continuous Time Hierarchical Co-Simulation." In ACM Student Research Competition (MoDELS). Saint Malo, France: ACM New York, NY, USA, 2016. *Hans suggested the initial idea and Cláudio wrote the paper. Then Hans reviewed.*
- GOMES, CLÁUDIO, Paschalis Karalis, Eva M. Navarro-López, and Hans Vangheluwe. "Approximated Stability Analysis of Bi-Modal Hybrid Co-Simulation Scenarios." In 1st Workshop on Formal Co-Simulation of Cyber-Physical Systems, 345–60. Trento, Italy: Springer, Cham, 2017. https://doi.org/10.1007/978-3-319-74781-1_24. *Cláudio came up with the idea. Then Cláudio and Paschalis worked on a draft of the paper. Eva and Hans reviewed.*
- GOMES, CLÁUDIO, Benoît Legat, Raphaël M. Jungers, and Hans Vangheluwe. "Stable Adaptive Co-Simulation: A Switched Systems Approach." In IUTAM Symposium on Co-Simulation and Solver Coupling. Darmstadt, Germany, 2017. *Cláudio and Benoît came up with the initial idea, and wrote the paper draft. Then Raphaël and Hans reviewed.*
- GOMES, CLÁUDIO, Benoît Legat, Raphaël Jungers, and Hans Vangheluwe. "Minimally Constrained Stable Switched Systems and Application to Co-Simulation." In IEEE Conference on Decision and Control. Miami Beach, FL, USA, 2018. *Cláudio and Benoît refined the idea from the previous paper, and Raphaël suggested the link with entropy. Cláudio and Benoît worked on the implementation and proof, while Raphaël and Hans reviewed.*
- GOMES, CLÁUDIO, Bart Meyers, Joachim Denil, Casper Thule, Kenneth Lausdahl, Hans Vangheluwe, and Paul De Meulenaere. "Semantic Adaptation for FMI Co-Simulation with Hierarchical Simulators." SIMULATION, 2018, 1–29. https://doi.org/10.1177/0037549718759775. *Bart, Hans, and Joachim, came up with the initial idea. Cláudio and Casper refined the idea for co-simulation. Bart worked on the syntax of the language. Cláudio, Casper, and Kenneth worked on the formal semantics of the language, and its implementation. Bart and Cláudio wrote the initial draft of the journal, which was then reviewed by Hans and Paul.*
- GOMES, CLÁUDIO, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. "Co-Simulation: A Survey." ACM Computing Surveys 51, no. 3 (April 2018): Article 49. https://doi.org/10.1145/3179993. *The initial suggestion to perform a survey came from Peter and Casper. Casper and Cláudio collected and classified the paper, and wrote the initial draft of the survey. David, Peter, and Hans, reviewed.*

- GOMES, CLÁUDIO, Casper Thule, Julien DeAntoni, Peter Gorm Larsen, and Hans Vangheluwe. "Co-Simulation: The Past, Future, and Open Challenges." In Symposium On Leveraging Applications of Formal Methods, Verification and Validation, Vol. 11246. Lecture Notes in Computer Science. Limassol, Cyprus: Springer Verlag, 2018. https://doi.org/10.1007/978-3-030-03424-5_34. *Peter came up with the idea of extending our previous work to a position paper. Cláudio, Casper, and Julien worked on the first draft. Hans and Peter reviewed.*
- Schweiger, Gerald, CLÁUDIO GOMES, Georg Engel, Irene Hafner, Josef Schoeggl, Alfred Posch, and Thierry Nouidui. "Functional Mock-up Interface: An Empirical Survey Identifies Research Challenges and Current Barriers." In The American Modelica Conference. Cambridge, MA, USA, 2018. *Gerald and Josef initiated the survey. Cláudio suggested some of the researchers to be contacted. Cláudio and Gerald worked on the questions that were to be asked in both stages of the study. Irene, Georg, Josef, Alfred and Thierry reviewed the questions. Gerald, Josef, and Georg, collected and processed the results. Gerald, Josef, Georg, and Cláudio worked on the draft of the paper, which was then reviewed by Alfred, and Thierry.*

Overview of Activities

During my PhD, I participated in a number of scientific activities that were (to some extent) related to my research. A (non-exhaustive) list is included here.

Organization of Scientific Activities

- Co-organized Training School on Multi-Paradigm Modeling for Cyber-Physical Systems, 18th-21st November 2018, Pisa, Italy. `http://mpm4cps.eu/trainingSchools/pisa2018`
- Part of Program Committee of 2nd Workshop on Formal Co-Simulation of Cyber-Physical Systems (CosimCPS), 26th June 2018, Toulouse, France. `https://sites.google.com/view/cosimcps18`
- Part of the Program Committee of Research, Innovation and Vision for the Future conference, to be held on 20-22 March 2019, Danang, Vietnam.

Teaching

- Exercises for "Modelling of Software-Intensive Systems" course at University of Antwerp (2015 - 2019);
- Lab sessions for "Model-Drive Engineering" course at University of Antwerp (2017 - 2019);
- Tutorial on Co-simulation, at Spring Sim conference, 2018: Gomes, Cláudio, Casper Thule, Peter Gorm Larsen, Joachim Denil, and Hans Vangheluwe. "Co-Simulation of Continuous Systems: A Tutorial." University of Antwerp, September 22, 2018. http://arxiv.org/abs/1809.08463.
- Tutorial on Co-simulation, at Summer Sim conference, 2018.
- Tutorial on Stability analysis for co-simulation, at CosimCPS workshop conference, 2018: Gomes, Cláudio, Casper Thule, Kenneth Lausdahl, Peter Gorm Larsen, and Hans Vangheluwe. "Demo: Stabilization Technique in INTO-CPS." In 2nd Workshop on Formal Co-Simulation of Cyber-Physical Systems, to be published. Toulouse, France: Springer, Cham, 2018.
- Co-authored publication for popular science magazine "EuroHeat&Power": Schweiger, Gerald, Cláudio Gomes, Irene Hafner, George Engel, Thierry Stephane Nouidui, Niki Popper, and Josef-Peter Schoggl. "Co-Simulation: Leveraging the Potential of Urban Energy System Simulation." EuroHeat&Power 15, no. I–II (2018): 13–16.

Participation in Scientific Activities

- Attended several meetings of the COST Action MPM4PCS, resulting in multiple publications with the participants.
- Attended, and presented at, the Spring Sim conferences 2017 and 2018.
- Presented at the MoDELS Conference, 2016.
- Presented at the first and second editions of the CosimCPS workshop 2017 and 2018.
- Presented at the IUTAM Symposium on Co-Simulation and Solver Coupling 2017
- Presented at the ISOLA Conference, 2018.
- Reviewed papers for:
  - Software Language Engineering journal;
  - Spring Simulation conference;
  - Winter Simulation conference;
  - Engineering w ith Computers journal;
  - CosimCPS Conference;
  - Machine Theory and Practice journal;
  - IFAC Workshop on Distributed Estimation and Control in Networked Systems;
  - Oil & Gas Science and Technology Journal;
  - Research, Innovation and Vision for the Future conference;
  - Simulation Modelling Practice and Theory journal; and
  - Software and Systems journal;

Chapter 1

Introduction

It is not just that systems are complex. Their development process is also complex. While systems comprised of many interacting, heterogeneous, components, are fundamental to our society [230, 276], we argue that the complexity in their development process should be reduced.

One can identify two main causes of complexity in the development process [351]:

Concurrency Concurrent engineering processes arise out of the need to deliver products faster [76]. However, they require frequent communication among different teams, to avoid late integration problems [104, 301, 351].

Specialization Specialization arises as our knowledge matures on each domain, opening new markets for supplier companies [93]. While there are clear benefits to specialization, the Original Equipment Manufacturer (OEM) has difficulties obtaining detailed descriptions of externally supplied components, due to Intellectual Property.

1.1 Motivation

Modeling and Simulation (M&S) techniques have proven to mitigate these issues [131, 321]:

- frequent integration can be attained if every model of the system is accessible in a single repository, so that every engineer can interact with the models and run simulations (e.g., see [193, 196]); and
- external suppliers can provide, or the OEM can build, abstract models of supplied components, based on well known and publicly accessible physical models.

However, M&S techniques are insufficient to enable the global optimization of the system design, because:

- simulations integrating multiple sub-models built with different tools become increasingly difficult, as these tools mature [359] (e.g., in Bosch at least 100 different M&S tools are used [45]));
- simulations comprising externally supplied components have limited fidelity [47]; and

- sub-system prototypes should be integrated with models of the rest of the system, to study the impact on their realization [18, 263].

Co-simulation is a response to the need to unlock the full potential of modeling and simulation techniques. It consists of the theory and techniques to enable global simulation of a coupled system via the composition of simulators [216]. Each simulator is broadly defined as a *black box* capable of exhibiting behavior, consuming inputs and producing outputs. Examples of simulators include dynamical models being integrated by numerical solvers [87], software and its execution platform [108], dedicated real-time hardware simulators (e.g., [180]), physical test stands (e.g., [378, Fig. 3]), or humans operating suitable interfaces (e.g., a steering wheel [92, Fig. 24], or graphical user interface [295, Fig. 6]).

1.2 Main Challenge and Contributions

Co-simulation shares the same fundamental challenge as any simulation-based technique: can the results be trusted? Naturally, if the results cannot be trusted, the utility of co-simulation is nil.

Our long term goal is to develop co-simulation to the stage where it can be applied seamlessly as part of engineering processes.

As such, the first part of our work is dedicated to understanding what makes co-simulation different than simulation (Chapters 3 and 4). The main conclusions are:

1. The co-simulation configuration space is much larger than the simulation one;
2. The current co-simulation standards are insufficient to enable correct co-simulation, because the black box nature of co-simulation offers little control over the implementation of simulators.
3. When the co-simulation results are incorrect, researchers propose solutions that require information about the simulators.
4. Developing stable co-simulation algorithms is a pre-condition to obtaining correct results.
5. Users of co-simulation often do not have the necessary information to correctly configure existing co-simulation algorithms.

The second part of our work focuses on identifying new use cases and scenarios where the results of the co-simulation are not correct, and the solutions proposed in the state of the art are insufficient to address them. In particular, our contributions are:

1. we show a first step to producing adaptive co-simulation algorithms that preserve the stability properties of the system (Chapter 5);
2. we provide an in-depth study of how state event detection in hybrid co-simulation can threaten the stability of the results, and offer a procedure to configure this detection, for a restricted class of systems (Chapter 6);
3. we develop the theory and techniques to control the configuration of simulators participating in co-simulations (Chapter 7); and
4. we propose a solution to improve the configuration of co-simulations, that leverages engineer's expertise in detecting incorrect results (Chapter 8), so that the co-simulation preserves the properties that are declared by engineers.

These contributions are related in the following ways:

- Contribution 1 and Contribution 2 both focus in fundamental research on preserving the stability property;
- Contribution 3 is used in the implementation of Contribution 4, and can potentially be used to apply Contribution 1 and Contribution 2, when these become more mature.

1.3 Limitations

Our contributions shed light into the many difficult problems that need to be addressed before our long term goal becomes a reality. However, it is hard to argue that we have solved each problem to the point where it no longer needs more research. In particular:

1. We assume that the IP of the supplier of sub-models is more protected if he shares the black box simulator, rather than if he shares the model. Where this assumption does not hold, there is still a need for co-simulation, as a way to integrate specialized M&S tools.
2. the body of knowledge in co-simulation is expanding at an unprecedented pace, so the state of the art survey will most likely be outdated by the time this work gets published;
3. Contribution 1 and Contribution 2 deal with NP-Hard and Undecidable problems, and we were able to apply them to very simple co-simulations only, where full information of the simulators is available;
4. Contribution 3 is limited to the current standards for co-simulation, hence it cannot fully change the configuration of each simulator; and
5. Contribution 4 does not guarantee that the configurations obtained are correct.

These are discussed in more detail in each chapter.

1.4 Structure

Chapter 2 introduces the main concepts in co-simulation, and scopes our work. The remaining chapters are self-contained, and each of them corresponds to one contribution.

Chapter 2

Background

In this chapter, we provide an overview of the main concepts in co-simulation, and the scope of our work. Many of the concepts informally defined here are refined in later chapters, to facilitate the description of the contributions.

We focus on human engineered systems, and their development processes. Human kind has been able to engineer and maintain complex systems for a long time. However, new technological advances and increasing expectations on these systems, lead to an increasing complexity not just in the systems, but also in their development process. We will call the first kind *complexity in the real world*, and the second kind *complexity in the simulation world*.

The next section details the causes of complexity in systems, and what the consequences are for the modeling and simulation activity. In particular, one of the conclusions is the need for model integration. Section 2.2 details how economic factors lead to development processes that make model integration more difficult. Then, Section 2.3 discusses how model integration can be done. Finally, Sections 2.4 to 2.6 introduce the main concepts in co-simulation, the model integration activity that we focus on.

2.1 Complexity in the Real World

We denote an engineered system as *original system*. Additionally, we consider systems that have a dynamic behavior, which may satisfy a set of properties of interest P, within a particular context. These properties are often represented as requirements that the behavior of the system has to satisfy. Failing to satisfy one of such properties may lead to catastrophic consequences. Example properties are: the system conserves energy, the system is continuous, the system's behavior remain within a particular region, etc.
Assumption 1. *Throughout this work, we assume that, for a given original system, the context for which it is to be used and the set of properties P, are fully specified.*

This assumption often does not hold in practice: unknown system/environment interactions can go undetected until the original system is in operation. A simple example of this is the deformation of a spring while operating under different temperatures. A more interesting

example is reported in [297], where a costly recipient tank was overfilled because the behavior of the system after the shutdown procedure was not taken into account.

The behavior of the system is observed by performing experiments. An experiment is a way to judge whether the system's behavior satisfies a subset of the properties P. We will denote the results of an experiment on system S by $[\![S]\!]$. Note that assumption 1 allows us to refer to the behavior of the system while assuming that such behavior was obtained within the correct context.

Definition 1 (Complex System)**.** We consider a system to be complex when it is difficult to partition the set of properties into multiple disjoint subsets of properties that can be studied individually. In other words, the satisfaction of P cannot be studied by a compositional study of disjoint subsets of P.

Definition 1 is consistent with the state of the art, where complex systems often have a large number of interacting, heterogeneous, components (structural complexity), and/or behaviors that are sensitive to external stimuli (dynamic complexity) [343].

Claim 1. *Existing works argue that a way to tackle the complexity in systems is to use models at the right level of abstraction, represented with the most appropriate formalism, to study a well defined set of properties (e.g., [86, 191, 359, 367, 368, 383]).*

Naturally, it takes time and resources to build models, hence Claim 1 needs to be taken as a rule of thumb. Nevertheless, it is instructive to understand its consequences when applied to complex systems.

Following the definitions of [219, 337], a *model* needs to be based on an original system, reflecting only a relevant subset of its properties, and can be used in place of the original, for some pre-defined purpose and within a particular context. This definition of model is intertwined with the definition of experiment: a model is created for the purpose of experimenting with it, and with a set of properties and context in mind [86, 383].

We denote by *dynamical models* those models whose purpose is to approximate the relevant behavior of the original system with respect to some properties of interest. They are characterized by a state and a notion of evolution rules. The state is a set of point values in a state space. The evolution rules describe how the state evolves over an independent variable, usually time.

The *behavior trace* $[\![M]\!]$ of a dynamic model M is the set of trajectories followed by the state (and outputs) of M. For example, a state trajectory $x \in [\![M]\!]$ can be defined as a mapping between a time base T and the set of reals $\mathbb{R}$, that is, $x : T \to \mathbb{R}$. We refer to the time variable $t \in T$ as *simulated time*—or simply *time*, when no ambiguity exists—defined over a time base T (typically the real numbers $\mathbb{R}$). Note the difference to the *wall-clock time* $\tau \in WcT$, which is the time that passes in the real world [135]. More details are given in Section 2.4. We need to postulate that every dynamical model has a behavior trace.

Definition 2. We say that a dynamic model M is *valid* w.r.t. system S when

$$\forall p \in P, [\![M]\!] \vDash p \implies [\![S]\!] \vDash p,$$

where P denotes the subset of properties that M was created to study, $[\![S]\!] \vDash p$ means that the behavior of S satisfies p, and $[\![M]\!] \vDash p$ means that the behavior of M satisfies p.

Assumption 2. *The models we work with are valid.*

In practice, it can be a challenge to recognize that a model is valid for a particular system and context [109, 383]. The simplest example of this is the model of a spring with Hooke's

law: it can only be used to measure the force of the spring for small deformations. A more interesting example is described in [336], where the authors ran a questionnaire through several experts in various domains of physics, asking them to identify the implicit assumptions in a simple model of a particle moving in a viscous medium. No expert was able to identify all the 29 assumptions, identified by their combined expertise.

Abstraction is a partial order between models with respect to a set of properties, commonly used in model checking [94].
Definition 3 (Abstraction)**.** Given a set of properties $P' \subseteq P$, a model M' is an abstraction of a model M w.r.t. P' if $\forall p \in P', [\![M']\!] \vDash p \implies [\![M]\!] \vDash p$.

In Definition 3, model M' will often be associated with a reduced context.

In Claim 1, a model M' is at the right level of abstraction with respect to a given P' if any other model M (comparable to M' with respect to the abstraction order), expressed in the same formalism, is not an abstraction of M' with respect to P'. Note that, due to time and resource constraints, it might not be possible to construct a model that is at the right level of abstraction.

The formalism of a model is defined here as the language used to represent the model. Formalisms have syntax and semantics. For example, ordinary differential equations is a formalism frequently used to create system models.

In Claim 1, the most appropriate formalism depends, at least, on:

Cognition (e.g., readability)
Tool Support (e.g., ease of use, collaboration support)
Library (e.g., most common mechanical system components)
Ecosystem (e.g., active community)

By these definitions, different sets of properties regulate the right level of abstraction, and the most appropriate formalism, for a model. For example, the best model to understand the temperature of a spring is different than the model required to understand the reaction force of the spring.

Therefore, applying Claim 1 to complex systems (Definition 1), one concludes that there seldom is a model, expressed in a single formalism and level of abstraction, that is suitable to study the whole set of properties P. Instead, the best model is typically a coupled model.
Definition 4 (Coupled Model)**.** A coupled model is combination of different models, interconnected in an architecture, expressed in possibly different formalisms, and at possibly different levels of abstraction, that share information.
Definition 5. Model integration is the act of computing the behavior trace of a coupled model. Model integration entails the formulation of the semantics of the coupled model.

For example, to understand whether a power window detects an obstacle and retracts without causing harmful deformation on the obstacle, one needs to construct a model that represents the dynamics of the window/motor sub-system, the dynamics of the software control sub-system, and the dynamics of how these sub-systems communicate (see Chapter 7). Prior research work has found [107] that a coupled model combining differential equations with state machines constitutes a good model to study this property.

2.2 Complexity in the Simulation World

Complexity in the simulation world is caused by the complexity of the original system, and the constraints on the development process, such as competitive pressures and specialization.

Traditionally, electromechanical systems were designed sequentially, i.e., the mechanical and structural components were designed first, then the electrical and electronic were selected or developed, then they were interconnected with the mechanical components, and so on [104]. This purely sequential development of components increases the product lead times [76]. Any attempt to optimize this process, and respond to competitive pressures, leads to concurrently developed components [230, 351]. When concurrently designed components need to be integrated, multiple problems can arise [104, 356, 359].

Increased specialization leads to distributed processes. As an anecdotal example, [93] presents the initial results of a study of product development in the world auto industry using data on passenger vehicle development projects from 20 automobile companies in Japan, Europe, and the US. They argue that, at least in the Japanese automotive industry, using components provided by specialized suppliers brings benefits to the development process. They also motivate the importance of concurrent development and specialization to reduce the critical path of projects.

To tackle increased specialization and distribution, the development process makes use of multiple models of the same system, developed concurrently, at different stages (e.g., the v-process [359]), each focusing on a subset of properties P.

The increased specialization makes it harder for each engineer to understand all the constructed models. This is where coupled models (Definition 4), and model integration, play a fundamental role: it allows engineers to quickly understand the impact of their changes in the system.

However, the distribution makes it difficult to construct models for externally supplied parts of the system, because these might constitute important intellectual property. For example, the system described in [297] comprises a water treatment sub-system that is built externally. How can one construct a coupled model of the system, using a sub-model that is provided by an external entity, with vested interest in not disclosing how the water treatment system works?

2.3 Model Integration

Models can be integrated in many different ways.
Example 1. For example, as shown in [266, 267], a state machine model can be integrated with a Causal Block Diagram (CBD) model to form a Hybrid Automata, or a Hybrid CBD. The Hybrid Automaton, formally defined in Chapter 6, is essentially an automaton where each state represents a mode of operation of the system, and within each mode, a CBD dictates the continuous behavior of the system, when it is in that mode. The Hybrid CBD is a CBD whose blocks can contain state machines, where each state dictates the output values of the block.

Co-simulation is a model integration approach where models are integrated using the

input/output mechanisms of each formalism. It has the potential to solve the challenges identified in Section 2.2:

- It enables the reuse of mature knowledge about how to build and simulate each model, thereby facilitating the integration of models created using specialized tools;
- It allows *black box* models to be integrated, thereby preserving the intellectual property in them.

In the following, we give some basic concepts on simulation, as a starting point to detail the co-simulation approach, the challenges it entails, and scope our contributions. These concepts are refined in later chapters.

2.4 Taxonomy of Dynamical Model Formalisms

Formalisms can be categorized according to:

Time Domain The time can be a continuous set (e.g., the set $\mathbb{R}$), discrete set (e.g., the set $\mathbb{N}$), or superdense set (e.g., the set $\mathbb{R} \times \mathbb{N}$)[1]. In Superdense time [232, 246, 247], each time point is a pair $(t, n) \in \mathbb{R} \times \mathbb{N}$.

State Domain The state domain can be a continuous set (e.g., $\mathbb{R}^n$, where $n \in \mathbb{N}$), a discrete set (e.g., $\mathbb{Q}^n$, where $n \in \mathbb{N}$), or a mix of both.

Behavior Trace The behavior trace can be discontinuous, continuous, differentiable.

Causality Models can be a-causal, when they can be coupled to other models without any notion of inputs and outputs, or causal, when outputs need to be connected to inputs and vice-versa.

Evolution The evolution of the state can be deterministic, stochastic, or non-deterministic.

This classification is based on the works of [157, 158, 191, 367].

Table 2.1 shows how the above taxonomy can be applied to classify multiple well known formalisms.

Table 2.1: Example classification of formalisms. ODE stands for Ordinary Differential Equations, DAE for Differential Algebraic Equations, and DEVS for Discrete Event System Specification.

Formalism	Time D.	State D.	Behavior T.	Causality	Evolution
ODEs	$\mathbb{R}$	$\mathbb{R}^n$	Differentiable	Causal	Determ.
DAEs	$\mathbb{R}$	$\mathbb{R}^n$	Differentiable	A-causal	Determ.
Difference Equations	$\mathbb{Q}$	$\mathbb{R}^n$	Discontinuous	Causal	Determ.
Petri-nets	$\mathbb{N}$	$\mathbb{N}^n$	Discontinuous	Causal	Non-Determ.
Automata	$\mathbb{N}$	$\mathbb{N}^n$	Discontinuous	Causal	Determ.
Differential Inclusions	$\mathbb{R}$	$\mathbb{R}^n$	Differentiable	Causal	Non-Determ.
Hybrid Automata	$\mathbb{R} \times \mathbb{N}$	Mix	Discontinuous	Causal	Non-Determ.
Classic DEVS	$\mathbb{R} \times \mathbb{N}$	Mix	Discontinuous	Causal	Determ.
Delayed ODEs	$\mathbb{R}$	$\mathbb{R}^n$	Differentiable	Causal	Determ.

[1] We exclude the singleton set for time because we consider dynamical models only.

The taxonomy introduced is not meant to be exhaustive, but rather give the reader a sense of the different kinds of formalisms being used, and how difficult it might be to integrate them in a co-simulation. For example, in [227], the authors report a successful integration of a railway traffic light Petri-net model with a CBD model. The result is a co-simulation that spans multiple parallel executions, branching whenever the Petri-net model exhibits non-deterministic state changes. However, the reported approach requires the CBD simulator to support parallel execution.

In this thesis, we will focus on the formalisms that are most commonly integrated in co-simulation practice: ODEs, DAEs, Automata, CBDs, and Hybrid Automata.

2.5 Simulation

There are two generally accepted ways of obtaining the behavior trace of a dynamical model:

Translational Translate the model into another model, which can be readily used to obtain the behavior trace. For example, obtaining the analytical solution of a set of linear differential equations.

Operational Use a solver – an algorithm that takes the dynamical model, and its inputs, as input, and outputs a behavior trace. For example, the application of a numerical solver to compute the solution to a set of differential equations.

Throughout this work we will focus on the latter approach.

A *simulator* (or solver) is an algorithm that computes the behavior trace of a dynamical model, given the inputs of the model. For some formalisms, the behavior trace can only be approximated. For example, if running in a digital computer, it is often the case that a simulator will only be able to approximate the trace of a set of differential equations. One can often create a model representing how the simulator approximates the behavior of the dynamical model (an example is shown in Section 3.7).

For a given model M and simulator $\mathcal{A}$, we denote the approximated behavior of the model by $[\![M]\!]_{\mathcal{A}}$. With this notation, the behavior trace computed is exact iff $[\![M]\!]_{\mathcal{A}} = [\![M]\!]$, and approximate otherwise.

In the case that $[\![M]\!]_{\mathcal{A}} \neq [\![M]\!]$, we define the error of the simulator as the abstract difference[2] between the original model and the induced model: $\|[\![M]\!] - [\![M]\!]_{\mathcal{A}}\|$, for some given norm $\|\cdot\|$. Based on the state of the art, we can distill the two unavoidable causes of errors in simulators: inability to compute unpredictable and continuous interactions between variables over time, and the finite representation of real numbers.

A simulator is accurate when $\|M - [\![M]\!]_{\mathcal{A}}\|$ is small enough to satisfy the following two conditions.

Definition 6 (Simulation Abstraction)**.** A simulator is a simulation abstraction when:

$$\forall p \in P, [\![M]\!]_{\mathcal{A}} \vDash p \implies [\![M]\!] \vDash p.$$

Definition 7 (Property Preservation)**.** A simulator is property preserving when:

$$\forall p \in P, [\![M]\!] \vDash p \implies [\![M]\!]_{\mathcal{A}} \vDash p.$$

[2] Abstract because we do not specify how to compute this difference.

To contrast accuracy with validity, note that validity is a property of a dynamical model whereas accuracy is a property of a simulator [87]. It is perfectly possible to have an accurate behavior trace of a model that is invalid, and vice versa.

Figure 2.1 summarizes these concepts, and their importance.

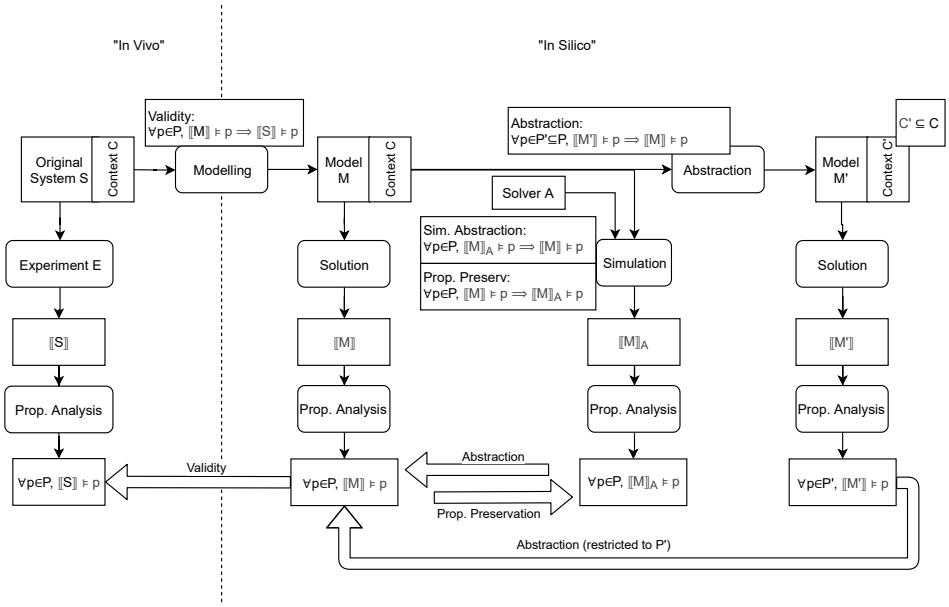


Figure 2.1: Summary of validity, abstraction, and property preservation, for simulation concepts. Validity is important because it enables conservative modeling. that is, when, for a valid model M, $[\![M]\!] \vDash p$ then we can conclude that $[\![S]\!] \vDash p$ without having to perform any physical experiment. Simulation abstraction is important because it frees the modeller from having to check $[\![M]\!] \vDash p$, while checking that $[\![M]\!]_{\mathcal{A}} \vDash p$ is often easier. Property preservation is important because, if $[\![M]\!]_{\mathcal{A}} \nvDash p$ then it informs the modeler that the problem is in the model ($[\![M]\!] \nvDash p$), and not in the solver.

One can apply the taxonomy identified in Table 2.1 to classify simulators as well. Additionally, we can identify one extra dimension:

Execution Time When computing the behavior trace of a dynamical model over an interval $[0, t]$ of simulated time, a computer takes τ units of wall-clock time that depend on t. The value τ can therefore be used to measure the run-time performance of simulators. Figure 2.2 highlights different kinds of simulation, based on the relationship between τ and t.

2.6 Co-simulation

We use the term *Simulation Unit (SU)* to denote something that produces a behavior trace, when inputs are provided. A SU can be a composition of a simulator and a dynamical model, or it can be a real-world entity (with appropriate interface). Notice that, in contrast

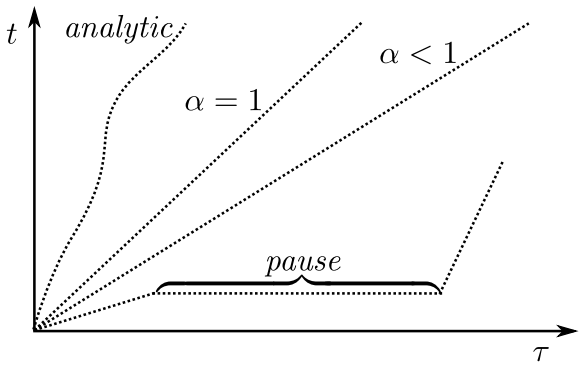


Figure 2.2: Classification of simulation with respect to execution time. Based on [268, 361]. In *real-time simulation*, the relationship between t and τ is $t = \alpha\tau$, for a given $\alpha > 0$. In most cases $\alpha = 1$ is required, but making sure this is obeyed by the simulation algorithm is one of the main challenges in real-time simulation, and by extension, of co-simulation. In as-fast-as-possible —or *analytical*— simulation, the relationship between τ and t is not restricted. Simulation tools that offer interactive visualization allow the user to pause the simulation and/or set a different value for α.

to a simulator, a SU only requires inputs to produce behavior. It represents an black box, a key concept in co-simulation.

A *simulation* is the behavior trace obtained with a SU. The correctness of a SU is dictated by the correctness of the simulation, which depends on the accuracy of the simulator and the validity of the dynamical model.

In co-simulation, the integration of models is done through SUs. These can be coupled via their inputs/outputs to produce a behavior trace of the coupled system. A *co-simulation*, a special kind of simulation, is the collection of combined simulations produced by the coupled SUs.

The SUs are coupled using a *master*, or orchestrator, algorithm. The master controls how the simulated time progresses in each SU and moves data from outputs to inputs according to a co-simulation scenario. A *co-simulation scenario* is the information necessary to compute a co-simulation. It includes how the values of each SU are shared with other SUs, the frequency of sharing, etc...

Analogously to the simulator and SU concepts, the composition of a specific master with a co-simulation scenario, yields a *co-SU*, which is a special kind of SU. It follows that a co-simulation is the simulation trace computed by a co-SU. This characterization allows hierarchical co-simulation scenarios, where co-SUs are coupled, and is compatible with the concepts of simulation introduced above:

Approximated Behavior Let M denote the coupled model. Then, a co-simulation unit $\mathcal{A}$ approximates the behavior of M. We will denote such behavior as $[\![M]\!]_{\mathcal{A}}$.

Property Preservation and Simulation Abstraction An accurate co-simulation algorithm satisfies the conditions in Definitions 6 and 7.

Kind of co-simulation unit The co-SU can be classified according to the taxonomy used to classify simulators.

Figure 2.3 summarizes these concepts.

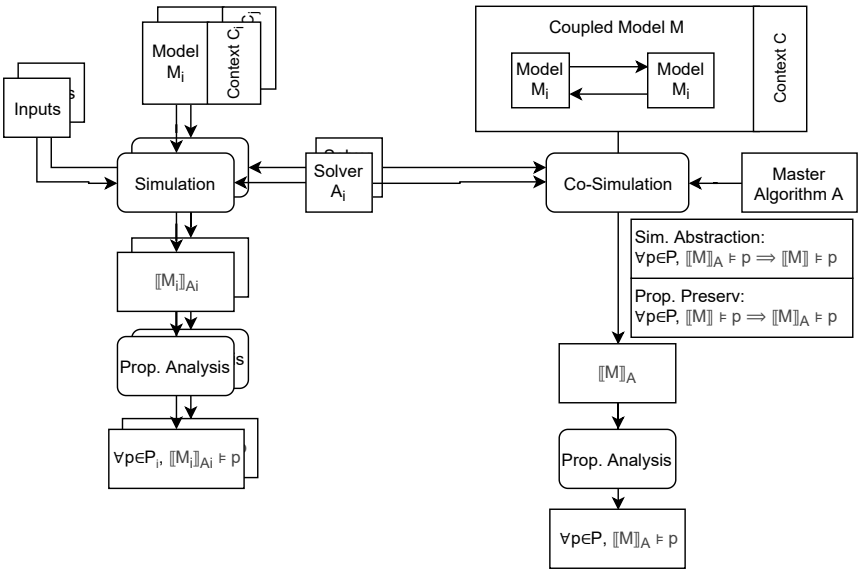


Figure 2.3: Summary of co-simulation concepts. The co-simulation makes use of each sub-model's solver, and the coupled model incorporates how the sub-models are connected. Also note that the inputs used for each sub-model's simulation are not necessarily the same as the inputs used in the co-simulation. Finally, there is no relationship between the properties satisfies by each sub-model, and the properties satisfied by the coupled model.

Chapter 3

State of the Art

Disclaimer The content in this chapter is adapted from:

- Gomes, Cláudio, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. "Co-Simulation: A Survey." ACM Computing Surveys 51, no. 3 (April 2018): Article 49. https://doi.org/10.1145/3179993.
- Gomes, Cláudio, Casper Thule, Julien DeAntoni, Peter Gorm Larsen, and Hans Vangheluwe. "Co-Simulation: The Past, Future, and Open Challenges." In Symposium On Leveraging Applications of Formal Methods, Verification and Validation, Vol. 11246. Lecture Notes in Computer Science. Limassol, Cyprus: Springer Verlag, 2018. https://doi.org/10.1007/978-3-030-03424-5_34.

In this chapter, we look at the state of the art in co-simulation. We give an historical overview of co-simulation, identify the main challenges and approaches to tackle such challenges.

The trend of historical data hints at a future with more and more virtual engineering, across all stages of the system's life-cycle. Furthermore, the proposed approaches share a common theme: they require more information about the black-box SUs, and/or demand that the SUs satisfy extra requirements. As such, we propose a taxonomy that structures these requirements and information, and charts the configuration space of the co-simulation.

This taxonomy is then used in a systematic literature survey, of papers from 2010 until 2015. The results of this classification help future standardization efforts, researchers, and simulator developers, understand the relative important of the different requirements.

3.1 Main Milestones

We summarize the main milestones of the past 30 years that lead to the modern uses of co-simulation. Figure 3.1 situates these in time, whose circled numbers are referenced throughout the text.

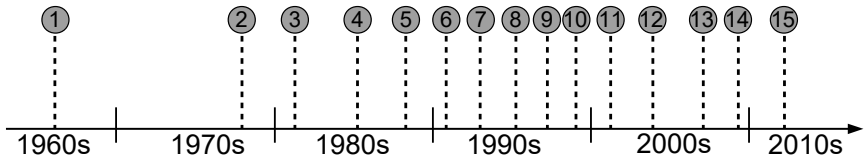


Figure 3.1: Timeline of co-simulation milestones. From 1970s up to 2015.

3.1.1 60s

(1) In the sixties, as parallel computer architectures matured, researchers explored how to parallelize traditional numerical solvers of differential equations (e.g., see [190, 255] and references thereof). From the early contributions, we can identify the following types of parallelism, relevant to simulation:

Algebraic the solution procedure to a set of algebraic equations (e.g., [91, 254]), or a single step of a numerical solver (e.g., [256]), is computed in parallel;

Time the simulation time interval is partitioned and the solution procedure is applied in parallel to the different intervals (e.g., [277])

State the state space is partitioned, and simulated in parallel (e.g., parallel partial differential equation solvers [82, 114])

The modern co-simulation approaches resemble the state parallel algorithms the most.

3.1.2 70s and 80s

(2) In the mid seventies, researchers (e.g., [38, 187, 291]) acknowledged that soil-structure and fluid-structure interaction problems should be partitioned into different sub-subsystems [124]. This is because their solution with a single numerical method was inefficient. These approaches where called partitioned methods [123], and they highlight the heterogeneity aspect in co-simulation.

To the best of our knowledge, the first discrete event synchronization algorithms were published in the late seventies [206], around the same time that Lamport [224] published his seminal paper regarding the ordering of events in distributed process networks. Discrete event simulators, described in Section 3.6, compute the behavior of a system by isolating the most important events and computing the state evolution of the system from one event to the next [134]. In this paradigm, a coupled system can be modeled as a set of sub-models that exchange events, which then are simulated in parallel, each in a separate process. The main challenge is the correct synchronization of the sub-models.

(3) In the late seventies and early eighties, as electrical circuits increased in size, their simulation algorithms were becoming a bottleneck in the development process because of the long simulation times. Practitioners noticed that, for sufficiently large circuits, only a small fraction of the subsystems had actively changing voltage levels, at any point in time. This led to the development of simulation techniques that, in a similar way to their discrete event based counterparts, only computed a new state of each subsystem when its outputs had changed significantly [190, 235, 252, 257, 275]. Additionally, to exploit parallel architectures and reduce numerical instabilities, the waveform relaxation techniques were introduced. These can be seen as a combination of state and time parallelism (recall

Section 3.1.1): during a computation interval $t \rightarrow t + H$, each subsystem was assigned to a simulator which approximated its solution in that interval, making a guess for the unknown input trajectories, using whatever simulation step size was required to keep the approximation error of that subsystem within tolerance. Then the simulators exchanged the computed solution trajectories, and were asked to re-compute the same interval, using the updated input trajectories.

These techniques made possible the simulation of large scale circuits because they naturally supported subsystems with different dynamics: systems which changed slowly where more quickly driven to convergence, and with larger simulation step sizes. They represent a parallel version of multi-rate solvers [140], and have been subject to extensive numerical analysis [250].

(4) In the late eighties, the release of Time Warp Operating System represented the optimistic facet in parallel discrete event simulation. It acknowledged that the performance of a parallel discrete event simulation could be increased by allowing the different processes to simulate as fast as possible, ignoring the absence of input events, and correcting causality violations. The corrections are made by rolling back the processes to a state that is consistent with the time of the input event that caused the violation (see Section 3.6).

(5) The performance of optimistic discrete event synchronization algorithms was such that it sparked the research into large scale simulations with humans interacting in realistic environments created by collaborating simulators. Developed during the 80s, SIMNET was dedicated to military training scenarios involving thousands of simulators representing, for instance, tanks or helicopters [253]. It encompasses an architecture and protocol to implement the optimistic synchronization of simulators in a distributed environment, with real-time constraints. In order to keep a reasonable level of accuracy and realism, one of the innovations is the concept of dead-reckoning models. A dead-reckoning model is a computationally lightweight version of some other model, whose purpose is to be used by interested simulators when there is a failure of communication, or when the synchronization times are far apart. A related approach is used in model predictive control [137].

3.1.3 90s

(6) Following the research into parallel algorithms, in the early nineties, process coordination languages emerged (e.g., Linda [46], Manifold [22]). Such languages can be used to manage process creation and communication [142], and therefore can be used to specify parallel synchronization schemes between simulator processes. During the same period, the *software architecture* research field proposed languages to abstract, structure, and reason about complex systems. One example is the Architecture Description Languages (ADLs) [138]. A ADL description specifies a system in terms of components and interactions among those components. Such languages helped 1) to clarify structural and semantics difference between components and interactions, 2) to reuse and compose architectural elements, 3) to identify/enforce commonly used patterns (e.g., architectural styles).

In 1990, United Airlines ordered 34 Boeing 777s, the first aircraft to be developed with concurrent engineering [193, 196]. The design was communicated fully in digital form, later aptly named a Digital Mockup Unit (DMU [21]), using CAD tools to showcase the different views of the system. This central repository of information served many purposes: (i) every team could consult the specifications of the subsystems made by any other team;

(ii) simulations could be carried out periodically, to detect problems in the design; (iii) both the assembly and maintenance phases of the system could affect the design phase, by running simulations of repairs and assembly.

This milestone represented an increase in the information that is taken into account for the design of the system. It now did not come only from requirements, but also from other stages of the life-cycle of the system: manufacturing, assembly and maintenance.

(7) As digital circuits became more complex, they comprised microprocessors running software. Rather than simulate the digital circuit, and deduce the execution of the software from there, researchers explored how to isolate the non-microprocessor part of the digital circuit, so that the software can be simulated directly. This field spawned the need for hardware/software co-simulation [90, 169, 315, 391]. Before using co-simulation, software developers had to develop their code with little information about the underlying hardware, leading to painful integration efforts later on. Later, they were able to quickly identify miscommunication errors before building hardware prototypes.

In the field of physical system simulation, researchers realized that there should be a standardized way to represent physical system models, so that these could be easily coupled to form complex systems [179, 218, 298, 369, 392]. One of these standards was called the Dynamical System Block (DSBlock) standard [287]. This proposal later inspired a widely adopted standard for co-simulation: the Functional Mockup Interface. (8) While the composition of DSBlocks still needs a solver, and is therefore not strictly considered co-simulation, this was a milestone in highlighting the need for standardization for continuous system co-simulation.

(9) As embedded systems were enhanced with communication capabilities, researchers noticed that the simulation of these distributed systems should not always be run at the same level of detail. Instead, the designers should be able to choose the level of detail they wanted for each embedded system: from the highest level of detail (circuit simulation), to the lowest (software simulation) [118, 181, 182, 209, 248, 320].

In the particular case of analog-digital co-simulation, each level of abstraction was solved by a different tool: a continuous time tool and a discrete event tool. The separation into continuous-time and discrete-event lead researchers to study some of the first hybrid co-simulation master algorithms [81, 125, 130, 354].

3.1.4 2000s

(10) The early 2000s was marked by multiple reported applications of co-simulation being used in industrial case studies (e.g., see [25, 124, 228] and Section 3.3). These had in common one aspect: two simulators were coupled, each specialized in one domain, in a feedback loop. (11) For example, in [25] the authors reports on the study of interactions between a pantograph (a mechanical structure on top of a train, connecting it to the electric grid), and a catenary (over hanging cable that transmits electricity to the train). A flexible body simulator was used to compute the behavior of the catenary, and a multi-body simulator was used for the pantograph. (13) In the meantime, the DIS standard, and its protocols, were generalized to non-real time applications, in what became the HLA (High Level Architecture) standard [14].

12 In order to ensure the correctness of coordinated heterogeneous model simulations, the Ptolemy and the Modhel'x projects proposed to expose some information about the behavioral semantics of languages (named Model of Computation) [60, 117]. Then, they defined adaptations so that they could be co-simulated. These works can be seen as a continuation of the process coordination languages that started in the 90s.

In the meantime, the domain of Hw/sw co-simulation matured with well defined Register Transfer Level (RTL) and Transaction Level Model (TLM) abstraction levels, which facilitated multi-abstraction co-simulation [37, 204, 312].

14 In 2008, the MODELISAR project published the FMI (Functional Mockup Interface) standard [47], whose essential contribution to co-simulation was the concept of Intellectual Property protection. It was an evolution of the DSBlock proposal, but recognizing that each subsystem might need its own simulator. This standard is widely adopted in industry[1] (see Chapter 4), where the simulation of externally supplied components can be costly due to high licensing costs.

Just as the adoption of simulation techniques in circuit design highlighted scalability into focus, so does the adoption of the FMI standard highlights this issue. The number of tools being used in the development of systems exploded. In the Bosch company, for example, there more than 100 different modeling and simulation tools [45] are in use.

Although there was some research about the coordination of black-box physical system simulators before the FMI Standard was published (e.g., [25, 165, 217]), it does not standardize the synchronization protocol between simulators. The main reason is that, as in continuous system simulation, there is no one-fits-all master algorithm. This is in contrast to discrete event simulation, where the implementations of the DIS and HLA standards provide everything to run the co-simulation.

Compared to the DMU initiative, the FMI represents a pragmatic approach that is aimed at being adopted by companies of different sizes, and does not require the intervenients to disclose their Intellectual Property, which is crucial in the interactions with suppliers [18, 359].

3.1.5 2010s

The current decade is marked by several applications of co-simulation across many domains (see Section 3.3) and across more development stages, with the Digital Twin [147] concept.

The application of co-simulation to simulate large scale distributed systems is discussed in [30, 31, 32, 136]. Furthermore, co-simulation is being proposed for early system validation and X-in-the-Loop studies, bringing hard real-time constraints to the set of challenges [115].

15 The Digital Twin extends the DMU concept not just to the design and assembly phases of the system, but also to the maintenance and operation. The essential idea is to use high fidelity models of the system, calibrated from sensory information collected during its

[1] `http://fmi-standard.org/`

operation, to affect how the system should operate, predict failures, schedule maintenance, etc... [56, 147, 163].

3.2 Emerging Trends and Challenges

Since co-simulation is but a special kind of simulation, the grand challenge is whether the results can be trusted. What makes co-simulation different is the fact that heterogeneous models are integrated. Obviously, if the results cannot be trusted, then the utility of the technique is nil.

Using the notations introduced in Chapter 2, let M denote the coupled model being co-simulated, $\mathcal{A}$ denote the co-simulation algorithm, and P the set of relevant properties. Then, the main challenge consists of satisfying property preservation (Definition 7):

$$\forall p \in P, M \vDash p \implies [\![M]\!]_{\mathcal{A}} \vDash p. \tag{3.1}$$

Assuming that co-simulation algorithms satisfy Equation (3.1), one can use the historical milestones to identify the trends, and take an informed guess of the challenges that ensue.

Throughout the history of co-simulation, the trend is a gradual shift towards the virtualization of not just the design of the system, but also assembly, operation, and maintenance.

The virtualization of the design has been one of the primary uses of co-simulation, backed up by concurrent engineering processes, such as the v-process [359].

The virtualization of the assembly is reflected by an increased demand in the information that should be taken into account at the design phase, with concepts like the Digital Mockup Unit (DMU).

Complex systems that need interaction with human operators require training interfaces. Marked by military simulators, the virtualization of operation refers to the creation of complex training environments at almost no cost by leveraging the same co-simulation scenarios used in the design phase.

Finally, extending the lifespan of systems, and reducing their downtime through the virtualization of their maintenance, is becoming a priority. This is in line with recent trends towards a circular economy, and a shift to an economy based on services, such as maintenance and disposal of the company's own products [355]. This means that co-simulation can be combined with advanced sensors to create smart monitors (also known as Digital Twins) that detect/predict failures, and therefore decrease the maintenance costs.

These trends allow us to identify the main context in which co-simulation is, and will be, used: Design Space Exploration, X-in-the-Loop, and Incremental Testing/Certification. Example applications of co-simulation that fit into these categories are described in Section 3.3. Each usage context is related to research topics, described below, and summarized in Figure 3.2.

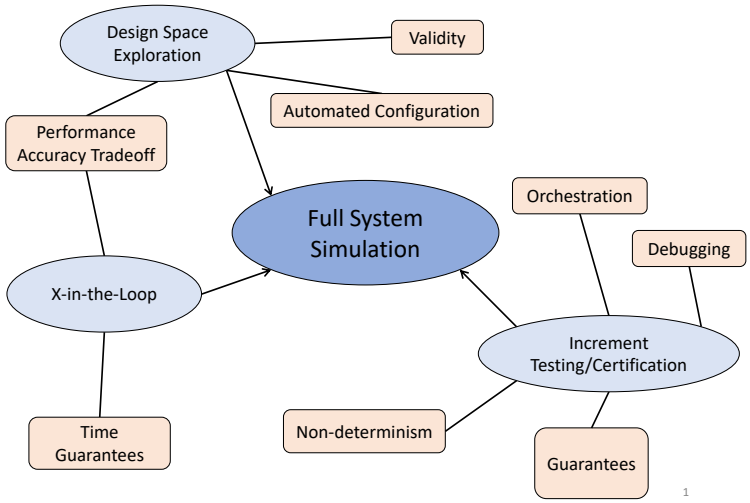


Figure 3.2: Overview of research topics in Co-simulation, according to the use cases.

3.2.1 Design Space Exploration

Design Space Exploration consists of the systematic analysis and evaluation of different designs over a parameter space. Co-simulation is used as part of the evaluation of a design. The following requirements are noteworthy:

Number of Designs The number of designs to be evaluated is large. Therefore, the co-simulations need to take as little wall clock time as possible.

Unsupervised Evaluation The experts do not typically inspect the behavior trace produced by the co-simulation algorithm. Therefore, the co-simulation needs to be automatically configured to produce trustworthy results.

Heterogeneous Designs The configuration of the co-simulation for one design can be wildly different from the configuration of the co-simulation for the other design. Additionally, the co-simulation should check the validity of each new coupled model. This is important because physical system models have many implicit assumptions, and their combination may violate those assumptions, purging their predictive value, making *validity* an important research topic.

3.2.2 X-in-The-Loop

X-in-the-Loop refers to co-simulations that are restricted in time and computing resources, due to the presence of human operators, animation requirements, or physical subsystems. In this context, there is a need for simulators which can provide contracts with timing and resource guarantees on their computation time, based on the inputs and parameters.

3.2.3 Incremental Testing/Certification

Incremental Testing/Certification consists of the co-simulation activities that are applied as part of concurrent engineering, where the models of each subsystem are refined as the development progresses and full system evaluation is run frequently. We highlight the need for co-simulations that provide formal guarantees on the accuracy. This could be done through some form of contract. Moreover, it should be possible to obtain an abstraction of each simulation unit that is appropriate to the kind of contracts defined. There is some research that could be used as a starting point [58, 83, 189, 245].

Once each simulator provides formal guarantees, then the master algorithm should ensure that the composition of those contracts, and other formal properties, can be satisfied. As highlighted by works on heterogeneous simulations and more recently in [237], the way to orchestrate the different simulators can lead to incorrect results. This is especially true when discrete models (with frequent and natural discontinuities) are in the loop since a minor change in timings can result in different behavior. This phenomenon is discussed in Chapter 6.

To illustrate, consider a simulator that guarantees there are no more than one discontinuity every 10 seconds. Then, depending on similar contracts satisfied by other simulators, a similar contract could be satisfied by the co-simulation.

One of the main use cases of the Digital Twin is to predict/detect failures, and analyze re-configurations of the system. This kind of what-if analysis, in an uncertain environment, is best done with a co-simulation that supports non-deterministic and stochastic models.

Finally, since co-simulation is meant to integrate black boxes, there is a need for research into how to detect faulty behavior, and understand its cause.

3.3 Co-simulation in Industry

This section attempts at providing an overview of the current usage of co-simulation in industry. Figure 3.3 summarizes the application domains in which there is a published report describing a co-simulation use, in the years 2011–2015. Each report is referenced in [157, Section 1.2].

The sub-sections describe three recent representative applications of co-simulation. Note that our description is based on published works only.

3.3.1 Exhaust Gas Recirculation (MAN Diesel & Turbo)

The work in [297] describes an exhaust gas re-circulation system, and a water handling system, schematized in Figure 3.4. The purpose is to clean and recirculate exhaust gas to a ship engine intake manifold. Due to new emissions legislation on NO_x, this system need to be improved. Since the development at MAN Diesel & Turbo (MDT) is split between different departments, tools, with limited sharing of models, co-simulation was applied to maximize reuse of models.

In the development of this system, the company initially used an in-house application framework, that simulated both the control system and the physical models of the ship

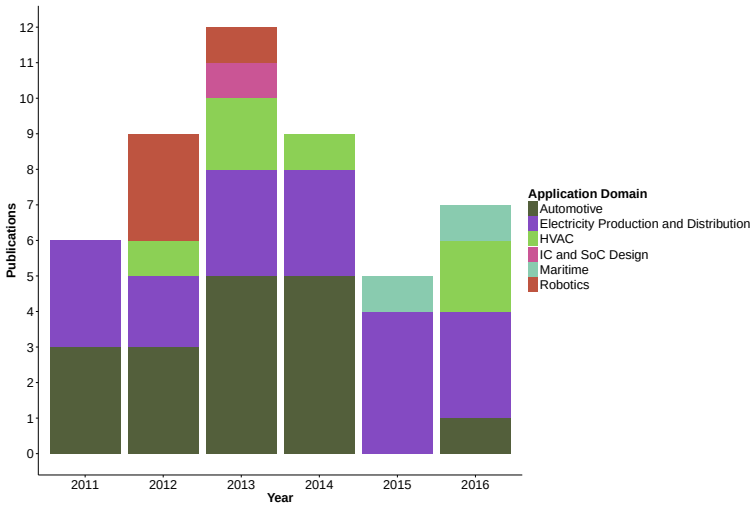


Figure 3.3: Research publications of co-simulation applications over 2011–2015. Taken from [157].

engine. However, this setup did not allow sufficiently detailed simulations of the physical models.

As a result, thanks to the FMI Standard, its support by MATLAB/Simulink®, and to the INTO-CPS co-simulation framework [350], the authors were able to combine the behavior of higher fidelity models, with the behavior of the controller under development, simulated by an in-house C++ software application framework.

Through co-simulation, it was possible to reproduce and correct an issue that was previously encountered only during a (costly) Hardware-in-the-loop simulation with a physical engine test bench available at the MDT research center in Copenhagen.

The authors believe that, had this approach been used from the start, then a water tank overflow problem could have been discovered before running the software on an expensive engine test bench.

3.3.2 Driverless Lawn Mower (AGROINTELLI)

Co-simulation has been applied to the development of a steering controller of an industrial size driverless lawn mower [127], developed by AGROINTELLI[2].

Besides aiding in the development of the control and navigation system of the lawn mover, co-simulation was used to investigate alternative designs, in a Design Space Exploration (DSE), that would otherwise be both costly and time-consuming to test with physical prototypes. Figure 3.5 shows example trajectories computed by alternative designs.

The co-simulation scenario consists of three parts: a simulator representing the vehicle dynamics, a simulator representing the control algorithm, and a simulator to convert values between the two (more about semantic adaptation in Chapter 7). During the DSE, each

[2] `http://www.agrointelli.com/`

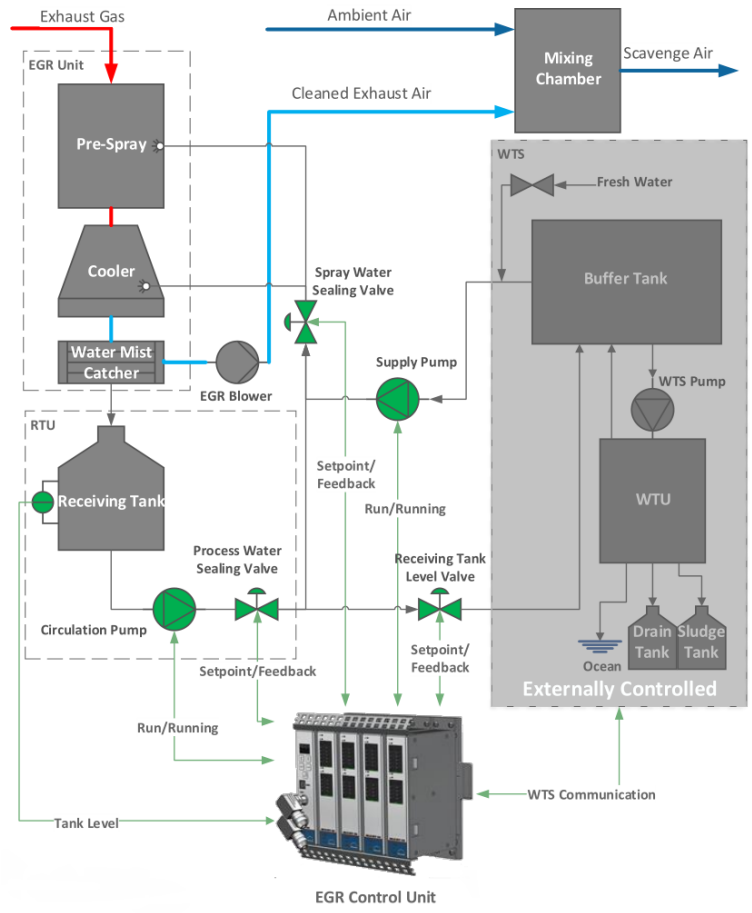


Figure 3.4: Exhaust Gas Re-circulation system. Taken from [297]. The exhaust gas is cleaned by spraying water into it, and allowing the mixture to cool down and flow into a receiving tank. Then, the (dirty) water is pumped to a water treatment center (externally developed) to be purified and reused.

alternative design was projected in a 3D animation based on the game engine Unity, so that it could be visually inspected by designers and clients. Additionally, it was possible for a human operator to control the virtual lawn mower with a joystick, highlighting the real-time constraints of the co-simulation.

Recognizing the validity challenge, the authors made sure the co-simulation results were valid and accurate by developing an initial prototype and comparing the actual behavior with the predicted one.

3.3.3 Motion Compensated Crane (ControlLab)

We highlight the design of a motion compensated crane [97], developed by Control-Lab.

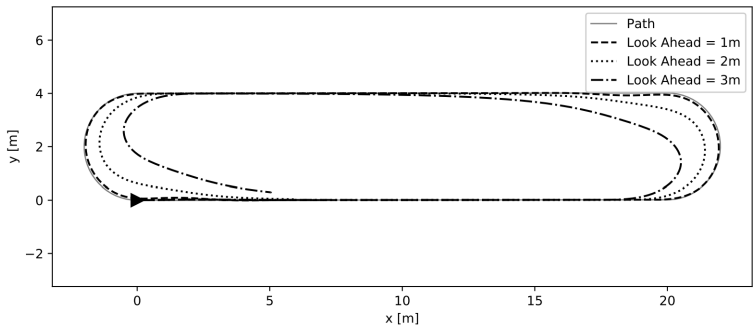


Figure 3.5: Simulated trajectories for look-ahead distance with velocity 1m/s. Taken from [127].

In this application, the models used in the development of the crane were reused in the development of a virtual reality environment. This environment was then used to safely train the crane operators. Figure 3.6 shows a third-person view of the operator using the controls on the crane to place himself on the platform.

Figure 3.6: 3D real-time simulation of a motion compensated crane. Taken from [97].

3.4 Co-simulation in Research

To assess the importance of co-simulation in the scientific community, we conducted a keyword analysis and searched for co-simulation related projects. We queried Scopus[3] for the keyword “co-simulation”. Figure 3.7 shows that the number of citations grew in an almost linear fashion from 2000 to 2017. As it can be seen in Figure 3.8, most of the publications can be assigned to the fields of Engineering (40%), followed by Computer Science (25%) and Mathematics (11%). Table 3.1 gives an overview of prominent recent research projects related to co-simulation.

This shows that there is an increasing interest in co-simulation as a research topic.

[3] www.scopus.com

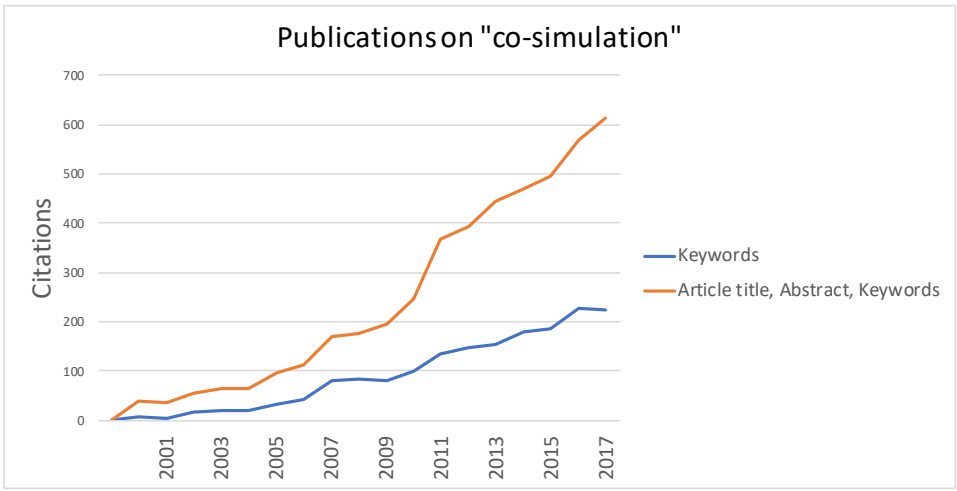


Figure 3.7: Publications that included the keyword “co-simulation”

3.5 Recent Survey Work

Co-simulation is now widely used in industry and researched in academia, having been applied in different domains. The extensive and scattered nature of this knowledge prompted researchers to initiate surveys of the state of the art. In the following, a short description of these efforts is provided.

3.5.1 Hafner and Popper

Hafner and Popper [171], recognizing that co-simulation has evolved in different domains, discuss the differences in terminology and attempt to classify co-simulation methods. The authors propose the following distinctions: (i) distinction by the state of development, (ii) by the field of application, (iii) by the model description, (iv) by numeric approaches, (v) by interfaces, and (vi) by multi-rate method.

Within state of development, one can find co-simulations that are run in order to integrate mature, independently developed simulation models, or co-simulation that are run as a divide-and-conquer way to speed up the computation. A traffic simulation model that is coupled with a weather prediction model, is an example of the former. The waveform relaxation, described in Section 3.1.2, is an example of the latter.

The partitioned methods, described in Section 3.1.2 represent an example of application of co-simulation due to different model descriptions.

The distinction by numerical approach corresponds to the different master algorithms that can be employed to synchronize the simulators. Examples of these are provided in Sections 3.6 and 3.7.

Distinction by interfaces focuses on the communication interfaces between the sub-models. These can be strong, characterized by the need for an implicit co-simulation approach, or weak, where an explicit co-simulation approach can be taken. More details on explicit and

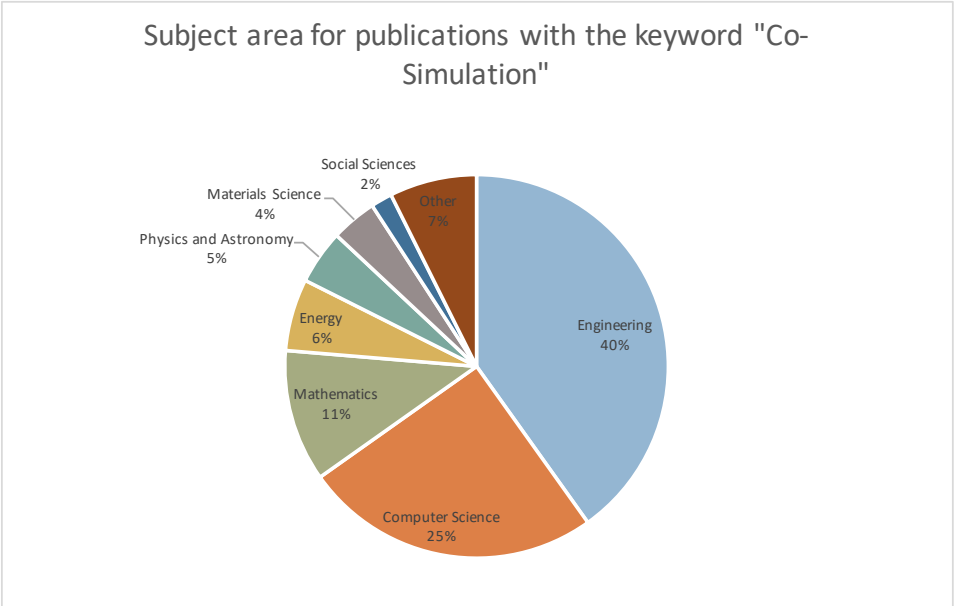


Figure 3.8: Subject area for publications that include the keyword “co-simulation”

implicit co-simulation approaches are given in Section 3.7.

The multi-rate distinction corresponds to the time stepping behavior of the simulators. This category comprises co-simulations that distinguish fast from slow simulators, executing them in parallel or in sequence.

3.5.2 Palensky et al.

Placing a focus on power systems, but still covering the main co-simulation algorithms (e.g., parallel/sequential, iterative/non-iterative, etc. . .), Palensky et al. [289] highlights the value of co-simulation for the analysis of the former. In a tutorial fashion, they go over the main concepts and challenges, providing a great introduction for new researchers in the field.

3.5.3 Our Survey

The above surveys complement our work. We define some common terminology in order to be more rigorous. Moreover, we focus on the main kinds of co-simulation, as identified in the systematic state of the art survey. No survey has identified all the types of co-simulation that we describe here, nor the requirements concerning the co-simulation units.

3.6 Discrete-Event-Based Co-simulation

From the historical overview, in Section 3.1, and the systematic literature review, we distinguish two major kinds of co-simulation: discrete event based, and continuous time

Table 3.1: Excerpt of research activities in the field of co-simulation in recent years.

Project	Duration	Goals
COSIBA [2]	2000–2002	Formulate a co-simulation backplane for coupling electronic design automation tools, supporting different abstraction levels.
ODETTE [9]	2000–20003	Develop a complete co-design solution including hardware/software co-simulation and synthesis tools.
MODELISAR [8]	2008–2011	Improve the design of embedded software in vehicles.
DESTECS [4]	2010–2012	Improve the development of fault-tolerant embedded systems.
INTO-CPS [7]	2015–2017	Create an integrated tool chain for Model-Based Design of CPS with FMI.
ACOSAR [1]	2015–2018	Develop a non-proprietary advanced co-simulation interface for real time system integration.
OpenCPS [10]	2015–2018	Improve the interoperability between Modelica, UML and FMI.
ERIGrid [6]	2015–2020	Propose solutions for Cyber-Physical Energy Systems through co-simulation.
PEGASUS [11]	2016–2019	Establish standards for autonomous driving.
CyDER [3]	2017–2020	Develop a co-simulation platform for integration and analysis of high PV penetration.
EMPHYSIS [5]	2017–2020	Develop a new standard (eFMI) for modeling and simulation environments of embedded systems.

based. Each has it's own essential characteristics and challenges. In this section, we focus on the former. Continuous time based co-simulation is described in Section 3.7.

The Discrete-Event-(DE)-based co-simulation approach describes a family of master algorithms and characteristics of SUs that are borrowed from the DE system simulation domain. We start with a description of DE systems, and then we extract the main concepts that characterize DE based co-simulation. These have a direct relationship with the concepts defined in Section 2.5.

Example 2. The traffic light is a good example of a DE model. It can be in one of three possible modes: `red`, `yellow`, `green`, or `off`. The off mode is often used by the police, which in some countries is characterized by a blinking yellow. Initially, the traffic light can be red. Then, after 60 seconds, it changes to green. Before those 60 seconds pass, a police officer may trigger a change from red to off. The output of this system can be an event signaling its change to a new color. There are no constraints regarding when the traffic signal can be interrupted. As such, the police officer can turn on and off the traffic light simultaneously.

This example captures some of the essential characteristics of a DE dynamical model: *reactivity* – instant reaction to external stimuli (turning off by an external entity); and *transiency* – a DE system can change its state multiple times in the same simulated time point, and receive simultaneous stimuli (police officer turns off and on the traffic light without any delay).

These characteristics are embraced in DE based co-simulation, where the master acknowledges that SUs can change their internal state and exchange values despite the fact that the simulated time is stopped.

3.6.1 DE Simulation Units

A DE SU is a black box that exhibits the characteristics of a DE dynamical model. Furthermore, it is typical to assume that DE SUs communicate with the environment via time-stamped *events*. This means that the outputs of SUs can be absent at times where no

event is produced.

We adapt the definition of the Discrete Event System Specification (DEVS [383]) to formally define a DE SU, and highlight the essential characteristics. Note that we chose this formalism due to its simplicity, having been described as the assembly language of DE models [366]. There are many other variants of DE formalisms. For instance, hardware description languages (VHDL and Verilog) and actor based systems (the DE director in Ptolemy II [308]).

Definition 8. We adopt the notation used in [365]. Let S_i denote a DE SU:

$$
\begin{aligned}
& S_i = \left\langle X_i, U_i, Y_i, \delta_i^{ext}, \delta_i^{int}, \lambda_i, ta_i, q_i(0) \right\rangle \\
& \delta_i^{ext} : Q_i \times U_i \to X_i \\
& \delta_i^{int} : X_i \to X_i \\
& \lambda_i : X_i \to Y_i \cup \{NaN\} \\
& ta_i : X_i \to \mathbb{R}_{\geq 0} \cup \infty \\
& q_i(0) \in Q_i \\
& Q_i = \{(x, e) | x \in X_i \text{ and } 0 \leq e \leq ta_i(x)\}
\end{aligned}
\tag{3.2}
$$

where:

- i denotes the SU reference;
- X_i, U_i, and Y_i are the set of possible discrete states, input events, and output events, respectively;
- $q_i(0)$ is the initial state;
- e denotes the elapsed units of time since the last transition (internal or external);
- $ta_i(x_i) \in \mathbb{R}$ is the time advance function that indicates how much time passes until the next state change occurs, assuming that no external events arrive;
- $\delta_i^{int}(x_i) = x_i'$ is the internal transition function that computes a new total state $(x_i', 0) \in Q_i$ when the current total state is $(x_i, ta_i(x_i)) \in Q_i$;
- $\delta_i^{ext}(q_i, u_i) = x_i'$ is the external transition function that computes a new total state $(x_i', 0) \in Q_i$ based on the current total state q_i and an input event u_i;
- $\lambda_i(x_i) = y_i \in Y_i \cup \{NaN\}$ is the output event function, invoked right before an internal transition takes place and NaN encodes an absent value;

The execution of a DE SU is described informally as follows. Suppose that the SU is at time $t_i \in \mathbb{R}_{\geq 0}$ and marks the current discrete state as x_i for $e \geq 0$ elapsed units of time. Since $e \leq ta_i(x_i)$, the total state is $(x_i, e) \in Q_i$. Let $tn = t_i + ta_i(x_i) - e$. If no input event happens until tn, then at time tn an output event is computed as $y_i := \lambda_i(x_i)$ and the new discrete state x_i is computed as $x_i := (\delta_i^{int}(x_i), 0)$. If, on the other hand, there is an event at time $ts < tn$, that is, u_i is not absent at that time, then the solver changes to state $x_i := (\delta_i^{ext}((x_i, e + ts - t_i), u_i), 0)$ instead.

If two events happen at the same time, both are processed before the simulated time progresses. Their processing order is determined by the DE Master (Section 3.6.2).

Due to the transiency and reactivity properties, the state and output trajectories of a DE SU can only be well identified using the superdense time base $\mathcal{T} \times \mathcal{N}$, introduced in Section 2.4. In this time base, a state trajectory is a function $x_i : \mathcal{T} \times \mathcal{N} \to V_{x_i}$, where V_{x_i} is the set of values for the state, and an output/input trajectory is $u_i : \mathcal{T} \times \mathcal{N} \to V_{u_i} \cup \{NaN\}$.

Simultaneous states and events can be formally represented with increasing indexes. See [67] for an introduction.

Example 3 shows instances of SUs, as defined in Definition 8.

A DE SU is passive: it expects some external entity to set the inputs and call the transition functions. This passivity enables an easier composition of SUs in a co-simulation, by means of a master algorithm, as will be shown later in Section 3.6.2. Algorithm 1 shows a master that computes the behavior trace of a single DE SU, as specified in Equation (3.2) with no inputs. Remarks: tl holds the time of the last transition; and the initial elapsed time satisfies $0 \leq e \leq ta_i(x_i(0))$;

If Algorithm 1 is used to coordinate the execution of the traffic light SU in Equation (3.3), then the resulting behavior trace is the piece-wise constant traffic light state $x_1(t)$, together with the output events. The latter is represented as a trajectory $y_i(t)$ that is absent everywhere except where an output is produced, according to ta_1.

ALGORITHM 1: Single autonomous DE SU orchestration.

Data: A $S_i = \langle X_i, \emptyset, Y_i, \delta_i^{ext}, \delta_i^{int}, \lambda_i, ta_i, (x_i(0), e_i) \rangle$.

1 $t_i := 0$;
2 $x_i := x_i(0)$; // Initial discrete state
3 $tl := -e_i$; // Account for initial elapsed time
4 **while** *true* **do**
5 $t_i := tl + ta_i(x_i)$; // Compute time of the next transition
6 $y_i := \lambda_i(x_i)$; // Output
7 $x_i := \delta_i^{int}(x_i)$; // Take internal transition
8 $tl := t_i$;
9 **end**

3.6.2 DE Co-simulation Orchestration

DE co-simulation scenarios are comprised of multiple DE SUs (Equation (3.2)) coupled through output to input connections, which map output events of one SU to external events in other SU.

Example 3. Consider the following DE SUs of a traffic light and a police office, respec-

tively:

$$
\begin{aligned}
&S_1 = \left\langle X_1, U_1, Y_1, \delta_1^{ext}, \delta_1^{int}, \lambda_1, ta_1, q_1(0) \right\rangle \\
&X_1 = Y_1 = \{red, yellow, green, off\} \\
&U_1 = \{toAuto, toOff\}\,; \qquad q_1(0) = (red, 0) \\
&\delta_1^{ext}((x_1, e), u_1) = \begin{cases} off & \text{if } u_1 = toOff \\ red & \text{if } u_1 = toAuto \text{ and } x_1 = off \end{cases} \\
&\delta_1^{int}(x_1) = \begin{cases} green & \text{if } x_1 = red \\ yellow & \text{if } x_1 = green \\ red & \text{if } x_1 = yellow \end{cases} \\
&\lambda_1(x_1) = \begin{cases} green & \text{if } x_1 = red \\ yellow & \text{if } x_1 = green \\ red & \text{if } x_1 = yellow \end{cases} \\
&ta_1(x_1) = \begin{cases} 60 & \text{if } x_1 = red \\ 50 & \text{if } x_1 = green \\ 10 & \text{if } x_1 = yellow \\ \infty & \text{if } x_1 = off \end{cases} \\
&S_2 = \left\langle X_2, U_2, Y_2, \delta_2^{ext}, \delta_2^{int}, \lambda_2, ta_2, q_2(0) \right\rangle \\
&X_2 = \{working, idle\} \\
&U_2 = \emptyset \\
&Y_2 = \{toWork, toIdle\} \\
&\delta_2^{int}(x_2) = \begin{cases} idle & \text{if } x_2 = working \\ working & \text{if } x_2 = idle \end{cases} \\
&\lambda_2(x_2) = \begin{cases} toIdle & \text{if } x_2 = working \\ toWork & \text{if } x_2 = idle \end{cases} \\
&ta_2(x_2) = \begin{cases} 200 & \text{if } x_2 = working \\ 100 & \text{if } x_2 = idle \end{cases} \\
&q_2(0) = (idle, 0)
\end{aligned}
\tag{3.3}
$$

With the following remarks:

- The current state of the model in the definition of δ_1^{ext} is $q_1 = (x_1, e)$ with e being the elapsed time since the last transition.
- The output event function λ_1 is executed *immediately before* the internal transition takes place. It must then publish the next state instead of the current.

To model a scenario where the police officer interacts with a traffic light, the output events Y_2 have to be mapped into the external events U_1 of the traffic light SU (Equation (3.3)). In Example 3, if $U_1 = \{toAuto, toOff\}$ are the external input events handled by the traffic

light SU, the mapping $Z_{2,1} : Y_2 \rightarrow U_1$ is defined by:

$$Z_{2,1}(y_2) = \begin{cases} toAuto & \text{if } y_2 = toIdle \\ toOff & \text{if } y_2 = toWork \end{cases} \tag{3.4}$$

This way, if the police officer changes to $working$ state at time tn, then the output signal $y_2 := toWork$ will be translated by $Z_{2,1}$ into an input event $u_1 := toOff$ of the traffic light SU.

Definition 9. Based on the idea of abstract SUs [386], we formalize a DE co-simulation scenario with reference cs as follows:

$$\langle U_{cs}, Y_{cs}, D, \{S_d : d \in D\}, \{I_d : d \in D \cup \{cs\}\}, \{Z_{i,d} : d \in D \wedge i \in I_d\}, \text{Select} \rangle \tag{3.5}$$

where:

- U_{cs} is the set of possible input events, external to the scenario;
- Y_{cs} is the set of possible output events from the scenario to the environment;
- D is an ordered set of SU references;
- For each $d \in D$, S_d denotes a DE SU, as defined in Equation (3.2);
- For each $d \in D \cup \{cs\}$, $I_d \subseteq (D \setminus \{d\}) \cup \{cs\}$ is the set of SUs that can influence S_d, possibly including the environment external to the scenario (cs), but excluding itself;
- For each $i \in I_d$, $Z_{i,d}$ specifies the mapping of events:

$$\begin{aligned} Z_{i,d} &: U_i \rightarrow U_d, \text{ if } i = cs \\ Z_{i,d} &: Y_i \rightarrow Y_d, \text{ if } d = cs \\ Z_{i,d} &: Y_i \rightarrow U_d, \text{ if } i \neq cs \text{ and } d \neq cs \end{aligned}$$

- Select : $2^D \rightarrow D$ is used to deterministically select one SU among multiple SUs ready to produce output events simultaneously, i.e., when at time t, the set of SUs

$$IMM(t) = \{d | d \in D \wedge q_d(t) = (x_d, ta_d(x_d))\} \tag{3.6}$$

has more than one SU reference. This function is restricted to select one from among the set $IMM(t)$, i.e., Select$(IMM(t)) \in IMM(t)$.

Example 4. The following co-simulation scenario cs couples the traffic light SU to the police officer SU:

$$\begin{aligned} &\langle \emptyset, Y_{cs}, \{1,2\}, \{S_1, S_2\}, \{I_1, I_2, I_{cs}\}, \{Z_{2,1}, Z_{1,cs}\}, \text{Select} \rangle \\ &Y_{cs} = Y_1; \qquad I_1 = \{2\}; \qquad I_2 = \emptyset; \qquad I_{cs} = \{1\}; \qquad Z_{1,cs}(y_1) = y_1 \end{aligned} \tag{3.7}$$

where: S_1 is the traffic light SU and S_2 the police officer SU (Example 3); Y_1 is the output of S_1; $Z_{2,1}$ is defined in Equation (3.4); and the omitted $Z_{i,d}$ functions map anything to absent (NaN).

The Select function is particularly important to ensure that the co-simulation trace is unique for a particular co-simulation scenario. For example, Example 4, and suppose that at time tn both SUs are ready to output an event and perform an internal transition. Should the traffic light output the event and perform the internal transition first, or should it be the

police office to do it first? In this example, the end result is the same but this is not the general case. In general, the order in which these output/transition actions are performed can lead to different co-simulation traces.

Algorithm 2 illustrates the master algorithm of an autonomous (without inputs) DE co-simulation scenario. It assumes that the co-simulation scenario does not expect external events, that is, all events that can affect the SUs are produced by other SUs in the same scenario. External output events are possible though. Remarks: t_{cs} holds the most recent time of the last transition in the scenario; e_d is the elapsed time of the current state $q_d = (x_d, e_d)$ of S_d; tn is the time of the next transition in the scenario; i^* denotes the chosen imminent SU; I_{cs} is the set of SUs that can produce output events to the environment; y_{cs} is the output event signal of the scenario to the environment; and $\{d|d \in D \wedge i^* \in I_d\}$ holds the SUs that S_{i^*} can influence.

ALGORITHM 2: Autonomous DE co-simulation scenario orchestration. Based on [365]

Data: A co-simulation scenario $cs = \langle \emptyset, Y_{cs}, D, \{S_d\}, \{I_d\}, \{Z_{i,d}\}, \text{Select} \rangle$.

$t_{cs} := 0$;
$x_i := x_i(0)$ for all $i \in D$; // Store initial discrete state for each unit
while *true* **do**
  $ta_{cs} := \min_{d \in D} \{ta_d(x_d) - e_d\}$; // Time until the next internal transition
  $tn := t_{cs} + ta_{cs}$; // Time of the next internal transition
  $i^* := \text{Select}(IMM(tn))$; // Get next unit to execute
  $y_{i^*} := \lambda_{i^*}(x_{i^*})$;
  $x_{i^*} := \delta_{i^*}^{int}(x_{i^*})$; // Store new discrete state
  $e_{i^*} := 0$; // Reset elapsed time for the executed unit
  if $i^* \in I_{cs}$ **then**
    $y_{cs} := Z_{i^*,cs}(y_{i^*})$; // Compute output of the scenario
  end
  for $d \in \{d|d \in D \wedge i^* \in I_d\}$ **do**
    $u_d := Z_{i^*,d}(y_{i^*})$; // Trigger internal units that are influenced by unit i^*
    $x_d := \delta_d^{ext}((x_d, e_d + ta_{cs}), u_d)$;
    $e_d := 0$;
  end
  for $d \in \{d|d \in D \wedge i^* \notin I_d\}$ **do**
    $e_d := e_d + ta_{cs}$; // Update the elapsed time of the remaining units
  end
  $t_{cs} := tn$; // Advance time
end

Figure 3.9 shows the behavior trace of the traffic light in the co-simulation scenario of Example 4.

Algorithm 2 is similar to Algorithm 1: i) The time advance of the scenario ta_{cs} corresponds to the time advance of a single SU; ii) The output produced by the state transition is analogous to the λ function of a single SU; and iii) The output and state transition of child S_{i^*}, together with the external transitions of the SUs influenced by S_{i^*}, are analogous to the internal transition of a single SU. It is natural then that a co-simulation scenario cs as

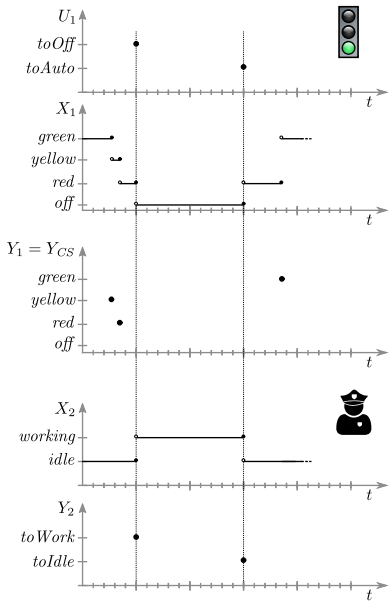


Figure 3.9: Example co-simulation trace of the traffic light and police officer scenario. Note that when the police interrupts the traffic light with the *toOff* event, no output is produced from the traffic light.

specified in Definition 9, can be made to behave as a single DE SU S_{cs}. Intuitively,

- the state of S_{cs} is the set product of the total states of each child DE SU;
- ta_{cs} is the minimum time until one of the DE SUs executes an internal transition;
- the internal transition of S_{cs} gets the output event of the imminent SU, executes the external transitions of all the affected SUs, updates the elapsed time of all unaffected SUs, and computes the next state of the imminent SU;
- the external transition of S_{cs} gets an event from the environment, executes the external transition of all the affected SUs, and updates the elapsed time of all the unaffected SUs [386].

The following definition formalizes this notion.

Definition 10. A co-SU is defined as:

$$
\begin{aligned}
&S_{cs} = \left\langle X_{cs}, U_{cs}, Y_{cs}, \delta_{cs}^{ext}, \delta_{cs}^{int}, \lambda_{cs}, ta_{cs}, q_{cs}(0) \right\rangle \\
&X_{cs} = \times_{d \in D} Q_d \\
&q_{cs}(0) = (\times_{d \in D} q_i(0), \min_{d \in D} e_d) \\
&ta_{cs}((\ldots, (x_d, e_d), \ldots)) = \min_{d \in D} \{ta_d(x_d) - e_d\} \\
&i^* = \text{Select}(IMM(t)) \\
&\lambda_{cs}(x_{cs}) = \begin{cases} Z_{i^*,cs}(y_{i^*}(tn)) & \text{if } i^* \in I_{cs} \\ NaN & \text{otherwise} \end{cases} \\
&\delta_{cs}^{int}(x_{cs}) = (\ldots, (x'_d, e'_d), \ldots), \text{ for all } d \in D, \text{ where:} \\
&\quad x_{cs} = (\ldots, (x_d, e_d), \ldots) \\
&\quad (x'_d, e'_d) = \begin{cases} (\delta_d^{int}(x_d), 0) \text{ if } i^* = d \\ (\delta_d^{ext}((x_d, e_d + ta_{cs}(x_{cs})), Z_{i^*,d}(\lambda_{i^*}(x_{i^*})), 0) \text{ if } i^* \in I_d \\ (x_d, e_d + ta_{cs}(x_{cs})) \text{ otherwise} \end{cases} \\
&\delta_{cs}^{ext}((x_{cs}, e_{cs}), u_{cs}) = (\ldots, (x'_d, e'_d), \ldots), \text{ for all } d \in D, \text{ where:} \\
&\quad x_{cs} = (\ldots, (x_d, e_d), \ldots) \\
&\quad (x'_d, e'_d) = \begin{cases} (\delta_d^{ext}((x_d, e_d + e_{cs}), Z_{cs,d}(u_{cs})), 0) \text{ if } cs \in I_d \\ (x_d, e_d + e_{cs}) \text{ otherwise} \end{cases}
\end{aligned}
\tag{3.8}
$$

Remarks:

- The discrete state of the co-SU is the Cartesian product of the total state of each child SU;
- The elapsed times of each child SU are managed solely by the co-SU, whenever there is a transition (internal or external);
- The external transition functions of each child are executed with the mapping of the events produced by the current state of the imminent child, and not the next one computed by $(\delta_d^{int}(x_d), 0)$;
- An internal transition of a child SU may cause an output event to the environment of the co-SU, if the child is connected to the output of the co-SU;
- A single internal transition causes not only a change in the child discrete state, but also, due to the child's output event, may cause external transitions in other child SUs. This is not a recursive nor iterative process: at most one external transition will occur in all the affected child SUs; if any of the affected SUs becomes ready for an internal transition, it waits for the next internal transition invoked from the coordinator of the co-SU;

The resulting co-SU S_{cs} behaves exactly as a DE SU specified in Equation (3.2). It can thus be executed with Algorithm 1 (in case of no inputs), or composed with other SUs in hierarchical co-simulation scenarios. Hierarchical co-simulation scenarios can

elegantly correspond to real hierarchical systems, a natural way to deal with their complexity [212].

In summary, DE based co-simulation exhibits the following characteristics:

reactivity: A DE SU (analogously, a DE co-SU) has to process an event at the moment it occurs.

transiency: In both Algorithm 2 and in a DE co-SU, the time advance ta_{cs} to the next imminent child internal transition can be zero for successive iterations, so an master has to tolerate the fact that simulated time may not advance for several iterations.

predictable step sizes: In a DE co-simulation scenario without inputs, the master, as shown in Algorithm 2, can always predict the next simulated time step. In a scenario with inputs, if the environment provides the time of the next event, then the next simulated time step can be predicted too. For this to be possible, black box DE SUs have to be able to inform the master what their time advance is. This is not a trivial task for DE SUs that simulate continuous systems whose future behavior trace, especially when reacting to future inputs, is not easily predicted without actually computing it.

The formalization presented thus far can be used to make explicit the main challenges in DE based co-simulation, along with the requirements their solutions impose. The following section focuses on these.

3.6.3 Technical Challenges

Causality

For the sake of simplicity, we presented Algorithm 2 as sequential. In a hierarchical co-SU, the imminent SU (closest to performing an internal transition) will be the one to execute, thus inducing a global order in the events that are exchanged. This order avoids causality violations but is too pessimistic. If an event $y_1(t_1)$ causes another event —by changing the internal state of some other SU, which in turn changes its next output event— $y_2(t_2)$, then $t_1 \leq t_2$, which is valid. However, the converse is not true: $t_1 \leq t_2$ does not necessarily imply that $y_1(t_1)$ has caused $y_2(t_2)$, which means that S_2 could execute before —in the wall-clock time sense— $y_1(t_1)$ without violating causality, at least within a small window of simulated time. To see why, suppose that S_1 and S_2 do not influence each other in the scenario. Then $y_2(t_2)$ would happen anyway, regardless of $y_1(t_1)$ occurring or not. Moreover, the co-simulation scenario holds information —the dependencies $\{I_d\}$— that can be used to determine who influences what [224][89].

A parallel optimistic master that takes $\{I_d\}$ into account is, in general, faster in the wall clock time sense, than a pessimistic, sequential one. However, most of these, the Time-warp algorithm [192] being a well known example, require rollback capabilities of SUs. This is because SUs proceed to advance their own time optimistically, assuming that any other SUs will not affect them, until they are proven wrong by receiving an event which occurs before their own internal time. When that happens, the SU has to rollback to a state prior to the time of timestamp of the event that just arrived. This may in turn cause a cascade of rollbacks in other affected SUs. Moreover, in parallel optimistic DE co-simulation, any of the SUs in the scenario needs (theoretically) to support multiple rollbacks and have enough memory to do so for an arbitrary distant point in the past [135]. This point in the past is

limited in Time-warp by the Global Virtual Time (GVT). The GVT represents the minimum internal time of all SUs. By definition, no event that is yet to be produced (in wall-clock time) can have a timestamp smaller than the GVT.

We make a distinction between multiple rollback and single rollback capabilities. To support single rollback, a SU needs to store only the last committed state, thereby saving memory.

Causality is a property worth preserving. If the original system is causal, so should the co-simulation be. Optimistic master algorithms ensure this by requiring rollback capabilities from child SUs, whereas pessimistic algorithms do so by ordering every event by its timestamp.

Determinism and Confluence

Determinism is also a property worth preserving. The Select function in Definition 9 is paramount to ensure deterministic behavior. This function cannot be defined arbitrarily. It must reflect the behavior of the real system.

The alternative to the Select function is to ensure that all possible interleavings of simultaneous event processing always lead to the same behavior trace – this is known as *confluence*. If a co-SU is confluent, then it is also deterministic.

Proving confluence is hard in general for black box DE co-simulation because it depends on knowledge about how the child SUs react to external events, which is potentially valuable IP. More research is needed to understand what information can be disclosed so that confluence can be ensured.

Dynamic Structure

Until now, the set of influencers $\{I_d\}$ for each SU d, in Definition 9, have been assumed to be fixed over time. From a performance perspective, a static sequence of dependencies may be too conservative, especially if used to ensure causality in optimistic parallel co-simulation. To see why, consider that in a large scale simulation, there is a SU S_1 which may influence SU S_2 but only under a very specific set of conditions, which may not be verified until a large amount of simulated time has passed. A pessimistic co-SU assumes that S_1 may always affect S_2 and hence, tries to ensure that the simulated time of S_2 is always smaller than S_1, to minimize possible rollbacks. This incurs an unnecessary performance toll in the overall co-simulation because S_1 does not affect S_2 most of the time. This is where making I_2 dynamic can improve the performance of the co-simulation since the co-SU will know that most of the time, S_1 does not affect S_2. Dynamic structure co-simulation allows for $\{I_d\}$ to change over time, depending on the behavior trace of the SUs. It can be used to study self-organizing systems [357][34].

Distribution

Co-SUs whose child SUs are geographically distributed are common [135]. Interesting solutions like computation allocation [269][362], bridging the hierarchical encapsulation [363], and the use of dead-reckoning models [229] have been proposed to mitigate the additional communication cost. Moreover, security becomes important, as pointed out, and addressed, in [280].

3.6.4 Standards for DE Co-simulation

Co-simulation presupposes that many different types of simulation tools can communicate with a master algorithm. To this end, establishing a common communication standard is crucial to avoid the combinatorial explosion of interfaces. Over the years, multiple standards were created for this purpose.

Discrete event based standards are proposed for simulations where events form a natural communication mechanism (for example, in software controllers, the state tends to evolve discontinuously, as a reaction to new inputs being transmitted by the environment). Continuous time based standards originate from differential equation based modeling activities, pervasive in many engineering domains. Hybrid based standards acknowledge that systems are comprised of both software and physical components, and therefore need to combine both events and continuous time interfaces. DEVS [383] can be considered one of the first discrete event based co-simulation standards. It standardizes not only the interface with which simulators communicate with the outside world, but also the master algorithm. Fueled by advances in parallel and distributed discrete event simulations, and their success in large scale military simulations, the Distributed Interactive Simulation (DIS) [188] standard was introduced as an evolution from SIMNET [253] (early 90s), later inspiring the creation of the High Level Architecture (HLA) standard [14] (early 2000s). The DIS standard targeted real-time distributed simulations, whereas HLA was targeting general purpose simulations.

3.6.5 Summary

The above concepts reflect the essential characteristics and challenges of DE-based co-simulation.

Event though we introduced an example dynamical model that is DE, Definition 8 can be used to approximate the behavior of a model that is not discrete event. This is discussed in Section 3.8.

In the next section, we focus on the other major kind of co-simulation: Continuous Time.

3.7 Continuous-Time-Based Co-simulation

In the continuous time (CT) based co-simulation approach, the master and simulation units' (SUs) behavior and assumptions are borrowed from the CT simulation domain. Each of these concepts, formally defined below, have a direct relationship with the concepts defined in Section 2.5.

3.7.1 CT Simulation Units

A CT dynamical model has a state that evolves continuously over time. It is easier to get the intuitive idea of this by an example.
Example 5. Consider a mass-spring-damper model, depicted in Figure 3.10. The state is

given by the displacement x_1 and velocity v_1 of the mass, and the evolution governed by:

$$\begin{aligned} \dot{x_1} &= v_1; & m_1 \cdot \dot{v_1} &= -c_1 \cdot x_1 - d_1 \cdot v_1 + F_e \\ x_1(0) &= p_1; & v_1(0) &= s_1 \end{aligned} \tag{3.9}$$

where $\dot{x}$ denotes the time derivative of x; c_1 is the spring stiffness constant and d_1 the damping coefficient; m_1 is the mass; p_1 and s_1 the initial position and velocity; and F_e denotes an external input force acting on the mass over time.

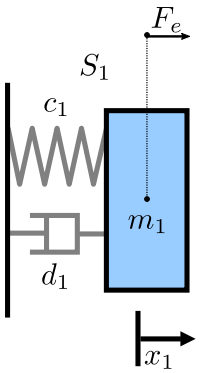


Figure 3.10: A mass-spring-damper system.

Equation (3.9) can be generalized to the state space form.
Definition 11. A dynamical model has the form:

$$\begin{aligned} \dot{x} &= f(x, u) \\ y &= g(x, u) \\ x(0) &= x_0 \end{aligned} \tag{3.10}$$

where x is the state vector, u the input and y the output vectors, and x_0 is the initial state.

We adopt Equation (3.10) to represent a dynamical model because these are the most commonly used models. These show up in the co-simulation of closed loop control systems, finite element codes, etc...

A solution $[x(t)]^T$ that obeys Equation (3.10) is the behavior trace of the dynamical model. In Example 5, the solutions $x_1(t)$ and $v_1(t)$ that satisfy Equation (3.9) constitute the behavior trace.

If f is linear and time-invariant, an analytical form for $x(t)$ can be obtained [28], which is an example of a behavior trace obtained via the translational approach, described in Section 2.5. In general however, the behavior trace can only be approximated. We describe one way this can be done.

If $f(x, u)$ is sufficiently differentiable, the vicinity of $x(t)$ can be approximated with a truncated Taylor series [87, 346]:

$$x(t + h) = x(t) + f(x(t), u(t)) \cdot h + \mathcal{O}\left(h^2\right) \text{ as } h \to 0, \tag{3.11}$$

where

$$\mathcal{O}\left(h^{n+1}\right) = \left(\lim_{h\to 0} \frac{x^{n+1}\left(\zeta(t^*)\right)}{(n+1)!} h^{n+1}\right) = \text{const } \cdot h^{n+1} \text{ as } h \to 0$$

denotes the order of the truncated residual term; $t^* \in [t, t+h]$; and $h \geq 0$ is the micro-step size. Equation (3.11) is the basis of a family of numerical solvers that iteratively compute an approximated behavior trace $\tilde{x}$.

For example, the forward Euler method is given by:

$$\begin{aligned} \tilde{x}(t+h) &:= \tilde{x}(t) + f(\tilde{x}(t), u(t)) \cdot h \\ \tilde{x}(0) &:= x(0) \end{aligned} \tag{3.12}$$

A CT SU behaves as a numerical solver computing a set of differential equations, as shown in the following example.

Example 6. For example, a SU S_1 of the mass-spring-damper, using the forward Euler solver, can be written by embedding the solver (Equation (3.12)) into Equation (3.9):

$$\begin{aligned} \tilde{x}_1(t+h_1) &:= \tilde{x}_1(t) + v_1(t) \cdot h_1 \\ \tilde{v}_1(t+h_1) &:= \tilde{v}_1(t) + \frac{1}{m_1} \cdot (-c_1 \cdot \tilde{x}_1(t) - d_1 \cdot \tilde{v}_1(t) + F_e(t)) \cdot h_1 \\ \tilde{x}_1(0) &:= p_1 \\ \tilde{v}_1(0) &:= s_1 \end{aligned} \tag{3.13}$$

where h_1 is the micro-step size, $F_e(t)$ is the input, and $[x(t+h), v(t+h)]^T$ is the output.

3.7.2 CT Co-simulation Orchestration

Example 7. Consider now a second system, depicted in Figure 3.11. It is governed by the differential equations:

$$\begin{aligned} \dot{x_2} &= v_2 \\ m_2 \cdot \dot{v_2} &= -c_2 \cdot x_2 - F_c \\ F_c &= c_c \cdot (x_2 - x_c) + d_c \cdot (v_2 - \dot{x_c}) \\ x_2(0) &= p_2 \\ v_2(0) &= s_2 \end{aligned} \tag{3.14}$$

where c_c and d_c denote the stiffness and damping coefficients of the spring and damper, respectively; and x_c denotes the displacement of the left end of the spring-damper. Combining Equation (3.14) with the forward Euler solver, yields the following SU:

$$\begin{aligned} \tilde{x}_2(t+h_2) &:= \tilde{x}_2(t) + \tilde{v}_2(t) \cdot h_2 \\ \tilde{v}_2(t+h_2) &:= \tilde{v}_2(t) + \frac{1}{m_2} \cdot (-c_2 \cdot \tilde{x}_2(t) - F_c(t)) \cdot h_2 \\ F_c(t) &= c_c \cdot (\tilde{x}_2(t) - x_c(t)) + d_c \cdot \left(\tilde{v}_2(t) - \dot{x_c}(t)\right) \\ \tilde{x}_2(0) &:= p_2 \\ \tilde{v}_2(0) &:= s_2 \end{aligned} \tag{3.15}$$

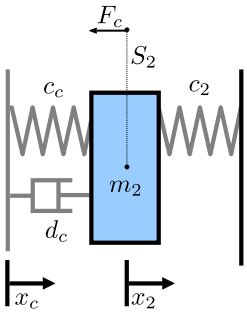


Figure 3.11: a mass-spring-damper system with a spring-damper connection.

where h_2 is the micro-step size, x_c and $\dot{x_c}$ are inputs, and F_c the output.

Example 8. Suppose the models in Examples 5 and 7 are coupled, by setting $x_c = x_1$, $\dot{x}_c = v_1$ and $F_e = F_c$. This results in the following coupled model, represented in Figure 3.12:

$$\begin{aligned}
\dot{x}_1 &= v_1 \\
m_1 \cdot \dot{v}_1 &= -c_1 \cdot x_1 - d_1 \cdot v_1 + F_c \\
\dot{x}_2 &= v_2 \\
m_2 \cdot \dot{v}_2 &= -c_2 \cdot x_2 - F_c \\
F_c &= c_c \cdot (x_2 - x_1) + d_c \cdot (v_2 - v_1) \\
x_1(0) &= p_1 \\
v_1(0) &= s_1 \\
x_2(0) &= p_2 \\
v_2(0) &= s_2
\end{aligned} \tag{3.16}$$

which can be written in the state space form (Equation (3.10)) as:

$$\begin{aligned}
\begin{bmatrix} \dot{x}_1 \\ \dot{v}_1 \\ \dot{x}_2 \\ \dot{v}_2 \end{bmatrix} &= \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\frac{c_1+c_c}{m_1} & -\frac{d_1+d_c}{m_1} & \frac{c_c}{m_1} & \frac{d_c}{m_1} \\ 0 & 0 & 0 & 1 \\ \frac{c_c}{m_2} & \frac{d_c}{m_2} & -\frac{c_2+c_c}{m_2} & -\frac{d_c}{m_2} \end{bmatrix} \begin{bmatrix} x_1 \\ v_1 \\ x_2 \\ v_2 \end{bmatrix} \\
\begin{bmatrix} x_1(0) \\ v_1(0) \\ x_2(0) \\ v_2(0) \end{bmatrix} &= \begin{bmatrix} p_1 \\ s_1 \\ p_2 \\ s_2 \end{bmatrix}
\end{aligned} \tag{3.17}$$

The objective of a co-simulation between the SU defined in Equation (3.13) and the SU defined in Equation (3.15) is to approximate the behavior of the coupled model in Example 8. In the following, we first convey the intuitive notions, and then provide the formal definitions.

In CT based co-simulation, to overcome the fact that each SU's micro-step sizes are independent, a communication step size H (also known as macro-step size or communication

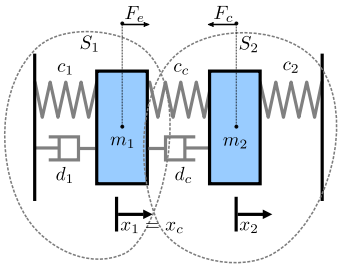


Figure 3.12: A multi-body system comprised of two mass-spring-damper subsystems.

grid size) has to be defined. H marks the times at which the SUs exchange values, and it is usually greater then or equal to all micro-step sizes[4].

Suppose a SU S_i is at time $n \cdot H$, for some natural n, and is asked by a master to execute until time $(n+1) \cdot H$. Before executing the co-simulation step, the unit needs to have values for its inputs. For each input variable, two scenarios can happen: S_i gets the input valued at $n \cdot H$, or valued at $(n+1) \cdot H$. If S_i gets the inputs valued at $n \cdot H$, then extrapolation must be used to estimate the input in any of the internal micro-steps of the SU. If, on the other hand, S_i gets the inputs valued at $(n+1) \cdot H$, interpolation must be used.
Remark 1. Each SU has to know the time stamp of each input value provided by the master. In the state of the art, to make the explanations clearer, this is frequently left implicit. However, the implementation of an SU must take this into account, as it determines the accuracy, stability, and the convergence rate of algebraic loops, of a co-simulation, as is discussed in Section 3.7.3. The implementation details of an SU are discussed in Chapter 7.
Definition 12. For a vector space U, we denote $U^{ts} = T^{|U|} \times U$ the time stamped vector space, for a time base T. We will use the notation $u(t)$ to represent the value u at timestamp $t \in T$.

Suppose the input vector space U_i of SU S_i is $U_i = \mathbb{R}$. Then, at time $n \cdot H + m \cdot h_i$, for $m \leq \frac{H}{h_i}$ and micro-step size h_i, an extrapolation function $\phi_{u_i}(m \cdot h_i, u_i(n \cdot H), u_i((n-1) \cdot H), \ldots)$, built from known input values, is used to approximate the value of $u_i(n \cdot H + m \cdot h_i)$. The importance of this approximation function cannot be overstated, as it determines the accuracy and stability of the co-simulation, as is discussed in Section 3.7.3. Analogously, an interpolation function is used when the master makes the input value available at time $(n+1) \cdot H$ before asking the SU to execute until time $(n+1) \cdot H$.
Example 9. The input F_e of the SU described in Equation (3.13) can be defined as:

$$F_e(n \cdot H + m \cdot h_1) := \phi_{F_e}(m \cdot h_1, F_e(n \cdot H), F_e((n-1) \cdot H), \ldots), \text{for } m \leq \frac{H}{h_1} \tag{3.18}$$

[4]While it is possible to have co-simulations where the communication step size is smaller than a micro-step size, this case is rare in practice, so we did not consider it. However, the results presented here are applicable to such co-simulations

Similarly, the inputs x_c and $\dot{x}_c$ of the SU described in Equation (3.15) can be defined as

$$\begin{aligned} x_c(n \cdot H + m \cdot h_2) &:= \phi_{x_c}(m \cdot h_2, x_c(n \cdot H), x_c((n-1) \cdot H), \ldots) \\ \dot{x}_c(n \cdot H + m \cdot h_2) &:= \phi_{\dot{x}_c}(m \cdot h_2, \dot{x}_c(n \cdot H), \dot{x}_c((n-1) \cdot H), \ldots) \\ &\text{for } m \leq \frac{H}{h_2} \end{aligned} \tag{3.19}$$

In the simplest case, the extrapolations can be constant. In the coupled mass-spring-dampers, this means:

$$\begin{aligned} \phi_{F_e}(t, F_e(n \cdot H)) &= F_e(n \cdot H); \\ \phi_{x_c}(t, x_c(n \cdot H)) &= x_c(n \cdot H); \\ \phi_{\dot{x}_c}(t, \dot{x}_c(n \cdot H)) &= \dot{x}_c(n \cdot H) \end{aligned} \tag{3.20}$$

In the state of the art, input approximation can be classified as:

1. Constant
2. Linear
3. Polynomial
4. Extrapolated-Interpolation [71, 116]
5. Context-aware [39, 40]; and
6. Estimated Dead-Reckoning Model [64, 338, 339]

Even though polynomial includes constant and linear approximations, we make the distinction between constant and linear approximations as these are the most commonly used in practice. See [19, 23, 71, 329] for an overview of linear and higher order extrapolation techniques and how these affect the accuracy of the co-simulation trace.

A possible master for the coupling described in Example 8, at a time $t = n \cdot H$, gets the outputs of both SUs and computes their inputs. Then, each SU is instructed to compute its behavior trace until the next communication step size, at $t = (n+1) \cdot H$, making use of the extrapolating functions described in Example 9 to get the inputs at each of the micro steps (Equations (3.18) and (3.19)).

Definition 13. We are now ready to formally define a CT SU S_i:

$$\begin{aligned} &S_i = \langle X_i, U_i, Y_i, \delta_i, \lambda_i, x_i(0), \phi_{U_i} \rangle \\ &\delta_i : \mathbb{R} \times X_i \times U_i^{ts} \to X_i \\ &\lambda_i : \mathbb{R} \times X_i \times U_i \to Y_i^{ts} \\ &x_i(0) \in X_i \\ &\phi_{U_i} : \mathbb{R} \times U_i^{ts} \times \ldots \times U_i^{ts} \to U_i \end{aligned} \tag{3.21}$$

where:

- X_i is the state vector space;
- U_i is the input vector space and U_i^{ts} its timestamped counterpart;
- Y_i is the output vector space;
- δ_i is the function that instructs the SU to compute a behavior trace from t to $t + H$, using the input approximation function ϕ_{U_i};
- $\lambda_i(t, x_i(t), u_i(t)) = y_i(t)$ is the output function; and
- $x_i(0)$ is the initial state.

Example 10. The SU in Equation (3.13) can be described as follows:

$$
\begin{aligned}
S_1 &= \left\langle \mathbb{R}^2, \mathbb{R}, \mathbb{R}^2, \delta_1, \lambda_1, \begin{bmatrix} p_1 \\ s_1 \end{bmatrix}, \phi_{F_e} \right\rangle \\
\delta_1(t, \begin{bmatrix} \tilde{x}_1(t) \\ \tilde{v}_1(t) \end{bmatrix}, F_e(t)) &= \begin{bmatrix} \tilde{x}_1(t+H) \\ \tilde{v}_1(t+H) \end{bmatrix} \\
\lambda_1(t, \begin{bmatrix} \tilde{x}_1(t) \\ \tilde{v}_1(t) \end{bmatrix}) &= \begin{bmatrix} \tilde{x}_1(t) \\ \tilde{v}_1(t) \end{bmatrix}
\end{aligned}
\tag{3.22}
$$

where $[\tilde{x}_1(t+H), \tilde{v}_1(t+H)]^T$ is obtained by the iterative application of the SU in Equation (3.13) over a finite number of micro-steps, making use of the extrapolation of F_e (defined in Equation (3.18)):

$$
\begin{aligned}
\begin{bmatrix} \tilde{x}_1(t+H) \\ \tilde{v}_1(t+H) \end{bmatrix} = \begin{bmatrix} \tilde{x}_1(t) \\ \tilde{v}_1(t) \end{bmatrix} &+ \begin{bmatrix} \dot{x}_1(t) \\ \dot{v}_1(t, \phi_{F_e}(t, F_e(t), \ldots)) \end{bmatrix} \cdot h \\
&+ \begin{bmatrix} \dot{x}_1(t+h) \\ \dot{v}_1(t+h, \phi_{F_e}(t+h, F_e(t), \ldots)) \end{bmatrix} \cdot h + \ldots
\end{aligned}
\tag{3.23}
$$

Definition 14. A CT co-simulation scenario with reference cs comprises the following information[5]:

$$
\begin{aligned}
&\langle U_{cs}, Y_{cs}, D, \{S_i : i \in D\}, L, \phi_{U_{cs}} \rangle \\
&L : (\Pi_{i \in D} Y_i) \times Y_{cs} \times (\Pi_{i \in D} U_i) \times U_{cs} \to \mathbb{R}^m
\end{aligned}
\tag{3.24}
$$

where D is an ordered set of SU references, each S_i is defined as in Equation (3.21), $m \in \mathbb{N}$, U_{cs} is the vector space of inputs external to the scenario, Y_{cs} is the vector space of outputs of the scenario, $\phi_{U_{cs}}$ a set of input approximation functions, and L induces the SU coupling constraints, that is, if $D = \{1, \ldots, n\}$, then the coupling is the solution to $L(y_1, \ldots, y_n, y_{cs}, u_1, \ldots, u_n, u_{cs}) = \bar{0}$, where $\bar{0}$ denotes the null vector. We choose to represent the coupling as the solution to an algebraic equation because the data exchange between the simulators can be more general than simply copying inputs to outputs.

Example 11. As an example, the co-simulation scenario representing the system of Figure 3.12 is:

$$
cs = \langle \emptyset, \emptyset, \{1, 2\}, \{S_1, S_2\}, L, \emptyset \rangle ; \qquad L = [x_c - x_1;\ \dot{x}_c - v_1;\ F_e - F_c]^T \tag{3.25}
$$

where:

- S_1 is the SU for the constituent system on the left (Equation (3.22)), and S_2 is the SU for the remaining constituent system;
- x_c, $\dot{x}_c$ are the inputs of S_2, and F_e is the input of S_1; and
- x_1, v_1 are outputs of S_1 and F_c is the output of S_2.

Algorithm 3 summarizes, in a generic way, the tasks of the master related to computing the co-simulation of a scenario cs with no external inputs. It represents the Jacobi communication approach: SUs exchange values at time t and independently compute the trace

[5]Please note that this formalization is related to the formalization proposed by [66], with the main differences: i) it is not designed to formalize a subset of the FMI Standard, ii) it accommodates algebraic coupling conditions, and iii) it does not explicitly define port variables.

until the next communication time $t + H$. The way the system in Equation (3.26) is solved depends on the definition of L. In the most trivial case, the system reduces to an assignment of an output $y_j(t)$ to each input $u_i(t)$, and so the master just gets the output of each SU and copies it onto the input of some other SU, in an appropriate order. Concrete examples of Algorithm 3 are described in [35, 73, 120, 132, 136, 165, 214, 375].

An alternative to the Jacobi communication approach is the Gauss-Seidel (a.k.a. sequential or zig-zag) approach, where an order of the SUs' δ function is forced to ensure that, at time t, they get inputs from a SU that is already at time $t + H$. Gauss-Seidel approach allows for interpolations of inputs, which is more accurate, but hinders the parallelization potential. An example of this algorithm is described in Chapter 7. Examples are described in [23, 25, 35, 358].

ALGORITHM 3: Generic Jacobi based master for autonomous CT co-simulation scenarios.

Data: An autonomous scenario $cs = \langle \emptyset, Y_{cs}, D = \{1, \ldots, n\}, \{S_i\}, L, \emptyset \rangle$ and a communication step size H.

Result: A co-simulation trace.

1 $t := 0$;

2 $x_i := x_i(0)$ for $i = 1, \ldots, n$;

3 **while** *true* **do**

4 Solve the following system for the unknowns:

$$\begin{cases} y_1 = \lambda_1(t, x_1, u_1(t)) \\ \ldots \\ y_n = \lambda_n(t, x_n, u_n(t)) \\ L(y_1, \ldots, y_n, y_{cs}, u_1, \ldots, u_n) = \bar{0} \end{cases} \tag{3.26}$$

$x_i := \delta_i(t, x_i, u_i(t))$, for $i = 1, \ldots, n$; `// Instruct each SU to advance`

6 $t := t + H$; `// Advance time`

7 **end**

Similarly to DE based co-simulation, a CT co-simulation scenario, together with a master, should behave as a (co-)SU of the form of Equation (3.21), and thus be coupled with other SUs, forming hierarchical co-simulation scenarios:

- the state of the co-SU is the set product of the states of the internal SUs;
- the inputs are given by U_{cs} and the outputs by Y_{cs};
- the transition and output functions are implemented by the master;
- the communication step size H used by the master is analogous to a SU's micro-step sizes, and
- the input extrapolation function is ϕ_{U_i}.

An example of this construction is given in Chapter 7.

Algorithm 3 makes it clear that the SUs can be coupled with very limited information about their internal details. In concrete:

- The output λ_i and state transition δ_i functions need to be executable but their internal details can remain hidden;
- the inputs u_i and their timestamps need to be accessible;

- the state variables can be hidden. These are represented merely to illustrate that the internal state of the SU changes when executing δ_i.

However, the *blind* coupling can lead to many problems, as will be discussed in the sections below. The common trait in addressing these is to require more from the individual SUs: either more capabilities, or more information about the internal (hidden) dynamical model.

3.7.3 Technical Challenges

In the following, to ease the notation, let $t_n = n \cdot H$.

Composition – Algebraic Constraints

In the co-simulation scenario described in Equation (3.25), the coupling condition L (Definition 14) translates into a set of assignments from outputs to inputs. This is because the inputs of the SU of the system in the left hand side of Figure 3.12 and the outputs of the SU of the system represented in the right hand side of the same picture can be connected directly, and vice versa. In practice, the SUs' models are not created with a specific coupling patterns in mind and L can therefore induce more complex coupling constraints.

Example 12. As an example, adapted from [324], consider the system coupled by a massless rigid link, depicted in Figure 3.13. The first subsystem is the same as the one in the left hand side of Figure 3.12 and its SU is in Equation (3.13). The second constituent system is governed by the following differential equations:

$$\begin{aligned} \dot{x}_3 &= v_3 \\ m_3 \cdot \dot{v}_2 &= -c_3 \cdot x_3 + F_c \\ x_3(0) &= p_3 \\ v_3(0) &= s_3 \end{aligned} \tag{3.27}$$

And the following SU:

$$\begin{aligned} \tilde{x}_3(t + h_3) &= \tilde{x}_3(t) + v_3(t) \cdot h_3 \\ \tilde{v}_3(t + h_3) &= \tilde{v}_3(t) + \frac{1}{m_3} \cdot (-c_3 \cdot x_3(t) + F_c(t)) \cdot h_3 \\ \tilde{x}_3(0) &= p_3 \\ \tilde{v}_3(0) &= s_3 \end{aligned} \tag{3.28}$$

The input to S_3 is the coupling force F_c, and the output is the state of the mass $[\tilde{x}_3, \tilde{v}_3]^T$. The input to S_1 is the external force F_e and the outputs are the state of the mass $[\tilde{x}_1, \tilde{v}_1]^T$.

In Example 12, there is a mismatch between inputs and outputs. The outputs $[\tilde{x}_1, \tilde{v}_1]^T$ of the first SU cannot be coupled directly to the input F_c of the second SU, and vice versa. However, the massless link restricts the states and inputs of the two SUs to be the same. Whatever the input forces may be, they are equal and opposite in sign. Hence, any master algorithm has to find inputs that ensure the coupling constraints are satisfied:

$$L = [\tilde{x}_1(t_n) - \tilde{x}_3(t_n); \quad \tilde{v}_1(t_n) - \tilde{v}_3(t_n); \quad F_e(t_n) + F_c(t_n)]^T = \bar{0} \tag{3.29}$$

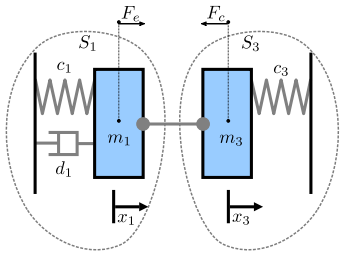


Figure 3.13: A multi-body system coupled by a mass-less link.

This problem has been addressed in [23, 25, 165, 166, 324, 329, 332]. The approach taken in [165] is worth mentioning because it defines a Boundary Condition Coordinator (BCC) which behaves as an extra SU, whose inputs are the outputs of the original two SUs, and whose outputs are F_e and F_c. They show that the initial co-simulation scenario with the non-trivial constraint can be translated into a co-simulation, with a trivial constraint, by adding an extra SU. This is illustrated in Figure 3.14.

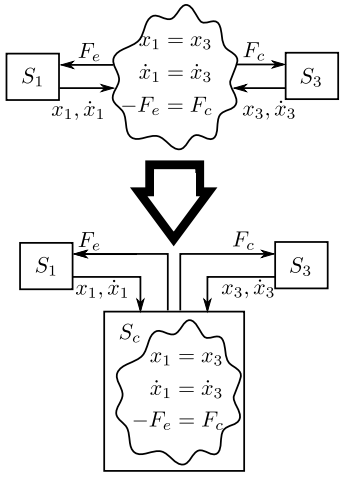


Figure 3.14: Transforming a co-simulation scenario with a non-trivial constraint into a simpler scenario by adding an extra SU that induces a trivial constraint. This promotes separation of concerns.

Transforming the co-simulation scenario to make it simpler marks an important step in separating the concerns of the master [150]. In fact, the newly created SU can be run with a smaller internal micro-step size, possibly to meet stability and accuracy criteria as shown in [165].

In many of the solutions proposed (e.g., [23, 25, 324, 329, 332]), information about the rate of change (or sensitivity) of outputs and states of each SU, with respect to changes in its inputs is required to solve the non-trivial coupling condition. This information can be either provided directly as a Jacobian matrix of the system and output functions, or estimated by finite differences, provided that the SUs can be rolled back to previous states.

A frequent characteristic of co-simulation: the availability of certain capabilities from SUs can mitigate the lack of other capabilities.

To show why the sensitivity information is useful, one of the tasks of the BCC is to ensure that $\tilde{x}_1 - \tilde{x}_3$ is as close to zero as possible, by finding appropriate inputs F_e and F_c. This is possible since $\tilde{x}_1$ and $\tilde{x}_3$ are functions of the inputs F_e and F_c, and $-F_e = F_c$. So the constraint can be written as

$$g(F_e) = \tilde{x}_1(F_e) - \tilde{x}_3(-F_e) = 0 \tag{3.30}$$

From one communication step to the next, g can be expanded with the Taylor series:

$$g(F_e(t_{n+1})) = g(F_e(t_n) + \Delta F_e) \approx g(F_e(t_n)) + \frac{\partial g(F_e(t_n))}{\partial F_e} \cdot \Delta F_e \tag{3.31}$$

From a known input $F_e(t_n)$, Equations 3.30 and 3.31 can be combined to obtain the input $F_e(t_{n+1})$ at the next communication step:

$$\begin{aligned} g(F_e(t_n) + \Delta F_e) \approx g(F_e(t_n)) + \frac{\partial g(F_e(t_n))}{\partial F_e} \cdot \Delta F_e = 0 \leftrightarrow \\ g(F_e(t_n)) = -\frac{\partial g(F_e(t_n))}{\partial F_e} \cdot \Delta F_e \leftrightarrow \\ \Delta F_e = -\left[\frac{\partial g(F_e(t_n))}{\partial F_e}\right]^{-1} \cdot g(F_e(t_n)) \leftrightarrow \\ F_e(t_{n+1}) = F_e(t_n) - \left[\frac{\partial g(F_e(t_n))}{\partial F_e}\right]^{-1} \cdot g(F_e(t_n)) \end{aligned} \tag{3.32}$$

with

$$\frac{\partial g(F_e(t_n))}{\partial F_e} = \frac{\partial \tilde{x}_1(F_e(t_n))}{\partial F_e} + \frac{\partial \tilde{x}_3(-F_e(t_n))}{\partial F_c} \tag{3.33}$$

A simple master algorithm will then perform the following steps, at each co-simulation step:

1. Let $\tilde{x}_1(nH), \tilde{x}_3(nH)$ be the current position outputs of the two SUs S_1 and S_3;
2. Perform a co-simulation step with a known F_e, obtaining $\tilde{x}_1^p(nH), \tilde{x}_3^p(nH)$ as new outputs.
3. Rollback SUs to state $\tilde{x}_1(nH), \tilde{x}_3(nH)$;
4. Pick a small ΔF_e, and perform a co-simulation step with $F_e + \Delta F_e$, obtaining $\tilde{x}_1^d(nH), \tilde{x}_3^d(nH)$;
5. Approximate $\frac{\partial g(F_e(t_n))}{\partial F_e}$ by finite differences and Equation (3.33);
6. Obtain a corrected F_e^c by Equation (3.32);
7. Rollback SUs to state $\tilde{x}_1(nH), \tilde{x}_3(nH)$;
8. Perform the final co-simulation step with F_e^c;
9. Commit states and advance time;

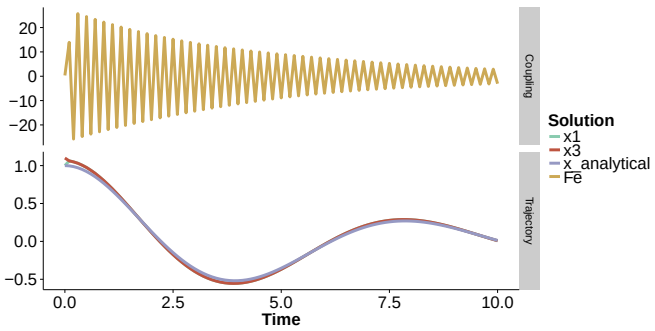


Figure 3.15: Co-simulation of algebraically coupled masses. Parameters are: $m_2 = 2, m_1 = c_1 = c_3 = d_1 = c_c = 1, H = 0.1, x_1(0) = 1.0, x_3(0) = 1.1, v_1(0) = v_3(0) = 0$. Notice the small disturbance at the initial conditions.

As can be seen in Figure 3.15, this coupling cannot be carried out without errors: the constraint $g(F_e(t_{n+1}))$ cannot be accurately forced to zero at first try. Furthermore, finding initial conditions and initial inputs that satisfy Equations 3.9, 3.27, and 3.29 is very important and usually requires a fixed point iteration. The above algorithm could be changed to perform an arbitrary number of iterations, repeating steps 1–7 until $g(F_e(t_{n+1}))$ is close enough to zero. This would increase the accuracy but also increase the amount of computation.

These examples show that rollback capabilities are important. If a SU is a black box, then the rollback capability has to be provided by the SU itself and there is little that the master can do to make up for the lack of the feature. If, on the other hand, the SU provides access to its state, and allows the state to be set, as in [48], then the master can implement the rollback by keeping track of the state of the SU. Rollback also plays a key role when dealing with algebraic loops in the co-simulation scenario.

Finally, to explain why this subsection refers to modular composition of SUs, the example in Figure 3.13 makes explicit one of the problems in co-simulation: the "rigid" and protected nature of SUs can make their coupled simulation very difficult. To contrast, in a white box approach where the equations of both constituent systems are available, the whole system is simplified, with the two masses being lumped together, and their coupling forces canceling each other out, as is shown in Example 13. The simplified system is a lumped mass-spring-damper, which is easily solvable. Such a symbolic approach is common in acausal modeling languages, such as Modelica [13].

Example 13. The coupled system is obtained by combining Equations 3.9, 3.27, and 3.29, and simplifying to:

$$\begin{aligned} \dot{x}_1 &= v_1 \\ (m_1 + m_3) \cdot \dot{v}_1 &= -(c_1 + c_3) \cdot x_1 - d_1 \cdot v_1 \\ x_1(0) &= p_1 \\ v_1(0) &= s_1 \end{aligned} \tag{3.34}$$

Figure 3.16 compares the behavior trace produced by Algorithm 3 when applied to the co-simulation scenario described in Equation (3.25), with the analytical solution, obtained

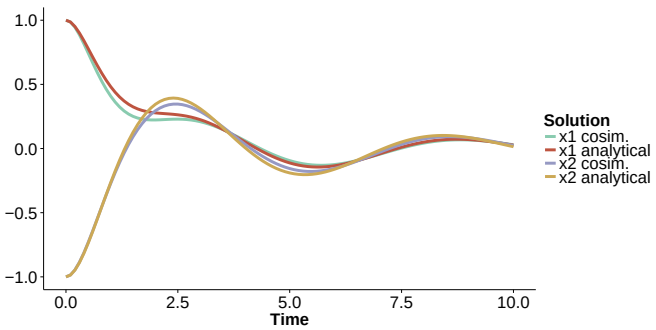


Figure 3.16: Comparison of co-simulation with co-modelling for the sample coupled system. Parameters are: $m_1 = m_2 = c_1 = c_2 = d_1 = c_c = 1, H = 0.1$.

from the coupled model of Equation (3.17). It is obvious that there is an error due to the extrapolation functions and the large communication step size $H = 0.1$.

Algebraic loops

Algebraic loops occur whenever there is a variable that indirectly depends on itself.
Example 14. To see how algebraic loops arise in co-simulation scenarios, supose two SUs, S_1 and S_2, are coupled in a feedback loop. As per Definition 13, the output of each S_i can be written as:

$$y_i(t) = \lambda_i(t, x_i(t), u_i(t)) \tag{3.35}$$

The SUs are coupled by a set of assignments from outputs to inputs, i.e.,

$$u_1(t) := y_2(t) \;\; ; \;\; u_2(t) := y_1(t) \tag{3.36}$$

where u_i is the input of SU S_i and y_j the output of a SU S_j, in the same co-simulation scenario. It is easy to see that the output of a SU may depend on itself. That is,

$$\begin{aligned} \mathbf{y_1}(\mathbf{t}) &= \lambda_1(t, x_1, u_1) \\ u_1 &= y_2 \\ y_2 &= \lambda_2(t, x_2, u_2) \\ u_2 &= \mathbf{y_1} \end{aligned} \tag{3.37}$$

We distinguish two kinds of algebraic loops [217]: the ones spanning just input variables, and the ones that include state variables as well. The first kind arises when the outputs of a SU depend on its inputs, while the second kind happens when implicit numerical solvers are used, or when the input approximating functions are interpolations. This distinction is important for the correct time synchronization of the SUs, as is shown in Chapter 7.
Example 15. To see how algebraic loops involving state variables arise, suppose that, in Example 14, ϕ_{u_i} is constructed from $u_i(t_{n+1})$:

$$u_i(t_n + m \cdot h_i) := \phi_{u_i}(m \cdot h_i, u_i(t_{n+1}), u_i(t_n), u_i(t_{n-1}), \ldots) \tag{3.38}$$

If an order can be imposed in the evaluation of the SUs that ensures $u_i(t_{n+1})$ can be computed from some $\lambda_j(t_{n+1}, x_j(t_{n+1}), u_j(t_{n+1})$ that does not indirectly depend on $u_i(t_{n+1})$,

then this approach —Gauss-Seidel— can improve the accuracy of the co-simulation, as shown in [23, 24, 25, 71, 201]. Obviously, the execution of SU S_i has to start after SU S_j has finished and its output $\lambda_j(t_{n+1}, x_j(t_{n+1}), u_j(t_{n+1})$ can be evaluated. If the input $u_j((n+1)$ depends indirectly on $u_i(t_{n+1})$, then an algebraic loop exists. The output function $\lambda_j(t_{n+1}, x_j(t_{n+1}), u_j(t_{n+1}))$ depends on the state of the SU at $x_j(t_{n+1})$, which in turn can only be obtained by executing the SU from time t_n to t_{n+1}, using the extrapolation of the input u_j, $\phi_{u_j}(m \cdot h_i, u_j(t_{n+1}, \ldots))$; any improvement in the input $u_j(t_{n+1}$, means that the whole co-simulation step has to be repeated, to get an improved $x_j(t_{n+1})$ and by consequence, an improved output $\lambda_j(t_{n+1}, x_j(t_{n+1}), u_j(t_{n+1}))$.

In Example 14, the input algebraic loops can be removed by replacing u_1 in Equation (3.35) by the corresponding extrapolation $\phi_{u_i}(H, u_i(t_{n-1}), \ldots)$ which does not depend on $u_i(t_n)$, thus breaking the algebraic loop.

As shown in [24, 217], and empirically in [35], neglecting a loop can lead to a prohibitively high error in the co-simulation. Instead, fixed point iteration technique should be used to solve algebraic loops. For those involving state variables, the same co-simulation step has to be repeated until convergence, whereas for loops over inputs/outputs, the iteration just repeats the evaluation of the output functions.

A master that makes use of rollback to repeat the co-simulation step with corrected inputs is called dynamic iteration, waveform iteration, and strong or onion coupling [184, 352]. If the SUs expose their outputs at every internal micro-step, then the waveform iteration can be used [235]. Strong coupling approaches are typically the best in terms of accuracy, but worst in terms of performance. A variant that attempts to obtain the middle-ground is the so-called semi-implicit method, where a fixed limited number of correction steps is performed. See [324, 329] for examples of this approach.

Until here, we have assumed full knowledge of the models being simulated in each SU to explain how to identify, and deal with, algebraic loops. In practice, with general black-box SUs, such knowledge is unavailable, and extra information is required to identify algebraic loops. According to [24, 44, 66], a binary flag denoting whether an output depends directly on an input is sufficient. A structural analysis, for example, with Tarjan's strong component algorithm [344], can then be performed to identify the loops.

Consistent Initialization of Simulators

The definition of a SU in Equation (3.21) assumes that an initial condition is part of the SU. However, as seen in Example 12, the initial states of the SUs can be coupled by algebraic constraints, through the output functions, which implies that the initial states of the SUs cannot be set independently of the co-simulation in which they are used. In the example, the constraint in Equation (3.29) has to be satisfied for the initial states:

$$\{\tilde{x}_1(0), \tilde{v}_1(0), \tilde{x}_3(0), \tilde{v}_3(0)\}\,.$$

In general, for a co-simulation scenario as defined in Equation (3.24), there is an extra coupling function L_0 that at the time $t = 0$, has to be satisfied. For example:

$$L_0(x_1(0), \ldots, x_n(0), y_1(0), \ldots, y_n(0), y_{cs}(0), u_1(0), \ldots, u_n(0), u_{cs}(0)) = \bar{0} \quad (3.39)$$

where:

- $x_i(0)$ denotes the initial state of S_i; and
- $L_0 : X_1 \times \ldots \times X_n \times Y_1 \times \ldots \times Y_n \times U_1 \times \ldots \times U_n \to \mathbb{R}^m$ represents the initial constraint, not necessarily equal to L in Equation (3.24).

Equation (3.39) may have an infinite number of solutions – as in the case of the example in Figure 3.13 – or have algebraic loops. The initialization problem (or co-initialization) is identified in [48] and addressed in [136].

Convergence – Error Control

In the context of co-simulation of CT systems, the most accurate trace is the analytical solution to the coupled model that the co-simulation scenario approximates. For example, the behavior of the coupled model in Equation (3.17), corresponding to the multi-body system in Figure 3.12, is approximated by the co-simulation scenario described in Equation (3.25). In practice, the analytical solution for a coupled model cannot be found easily, therefore calculating the error precisely is impossible for most cases. However, it is possible to get an estimate of how the error grows.

We now review the error control techniques in simulation, as these are the basis for the analogous techniques in co-simulation.

In simulation, the factors that influence the error are [87]: the model, the solver, the micro-step size, and the size of the time interval to be simulated.

For example, when the forward Euler solver (Equation (3.12)) is used to compute the approximated behavior trace of the dynamical model in Equation (3.10), in a single micro step, it is making an error in the order of

$$\left\| \underbrace{\left(x(t) + f(x(t)) \cdot h + \mathcal{O}\left(h^2\right)\right)}_{\text{by infinite Taylor series}} - \underbrace{\left(x(t) + f(x(t)) \cdot h\right)}_{\text{by forward Euler}} \right\| = \mathcal{O}\left(h^2\right)$$

Obviously, the order in the error made at one step $\mathcal{O}\left(h^2\right)$, most commonly called the local error, depends on:

- f having no unbounded derivatives – to see why, observe that if the derivative of f is infinite, then the residual term cannot be bounded by a constant multiplied by h^2. Fortunately, since most CT dynamic systems model some real system, this assumption is satisfied.
- The solver used – other solvers, such as the midpoint method, are derived by truncating higher order terms of the Taylor series. For the midpoint method, the local truncation error is $\mathcal{O}\left(h^3\right)$;
- Naturally, the larger the micro step size h is, the larger the local error $\mathcal{O}\left(h^2\right)$ is.

The local error assumes that the solver only made one step, starting from an accurate point $x(t)$. To compute the approximate behavior trace, the only accurate point the solver starts from is the initial value $x(0)$. The rest of the trace is approximate and the error gets compounded over the multiple steps. We denote such it by global error. For the forward Euler method, if there is a limit to how f reacts to deviations on its parameter $\tilde{x}(t) = x(t) + e(t)$ from the true parameter $x(t)$, that is, if

$$\|f(x(t)) - f(x(t) + e(t))\| \leq \text{const} \cdot e(t) \tag{3.40}$$

and const $< \infty$, then the order of the global error can be defined in terms of the micro-step size. This condition is called global Lipschitz continuity [121]. For the forward Euler solver, the global error is $\mathcal{O}(h)$.

For a solver to be useful, it must be convergent, that is, the computed trace must coincide with the accurate trace when $h \to 0$ [373]. It means the error can be controlled by adjusting the micro step size h.

In co-simulation, the error is influenced by the same factors as simulation, plus: the extrapolation functions, the master algorithm, and the co-simulation step size H. The extrapolation functions introduce error in the inputs of the SUs, which is translated into error in the state/outputs of these, causing a feedback on the error that can increase over time. Intuitively, the larger the co-simulation step size H, the larger is the error made by the extrapolation functions.

The same concepts (Local/Global Error and Convergence) apply to co-simulation, and we refer to [24] for calculations of the global order of the error.

The local error vector, in a co-simulation, is defined as the deviation from the analytical trace after one co-simulation step H, starting from an accurate point.

$$\begin{gathered} x_1(t+H) - \tilde{x}_1(t+H) \\ \cdots \\ x_n(t+H) - \tilde{x}_n(t+H) \\ y_1(t+H) - \tilde{y}_1(t+H) \\ \cdots \\ y_n(t+H) - \tilde{y}_n(t+H) \end{gathered} \tag{3.41}$$

where $\tilde{x}_i(t+H) = \delta_i(t, x_i(t), \phi_{u_i}(t))$, $\tilde{y}_i(t+H) = \lambda_i(t, \tilde{x}_i(t+H), \phi_{u_i}(t+H))$, and $x_i(t+H)$ and $y_i(t+H)$ are the true state vectors and outputs, respectively, for SU S_i.

To ensure that a co-simulation is convergent, contrarily to what Equation (3.40) might hint at, it is not enough that every SU is Lipschitz continuous. According to [24, 25, 73, 170, 217], the SUs need to be Lipschitz continuous, and the coupled model induced by the scenario coupling conditions needs to be of the form of Equation (3.10). This result only applies to the most common master algorithms (Jacobi, Gauss-Seidel, or Strong coupling) and polynomial input approximation techniques. Presence of algebraic loops, or complex coupling constraints, are factors that may make it impossible to write the coupled model in state space form [23].

For a convergent co-SU, some of the techniques traditionally used in simulation, have been applied in co-simulation to *estimate* the error during the computation:

Richardson extrapolation: This well-known technique is compatible with black-box SUs as long as these provide rollback and state saving/restore capabilities [24, 26, 136]. The essential idea is to get an estimate of the local error by comparing $[\tilde{x}_i(t+H), \tilde{y}_i(t+H)]^T$ with a less accurate point $[\bar{x}_i(t+H), \bar{y}_i(t+H)]^T$. The less accurate point can be computed by the same master but using a larger communication step size. We have seen that larger communication step sizes affect the accuracy so if the two points are not too far apart, it means the communication step

H does not need to be changed. It is importance to notice that the less accurate point $[\bar{x}_i(t+H), \bar{y}_i(t+H)]^T$ has to be computed from the accurate starting point $[\tilde{x}_i(t), \tilde{y}_i(t)]^T$.

Multi-Order Input Extrapolation: The outputs of two different order input approximation methods are compared [73, 75].

Milne's Device: Similar to the previous ones, but the extrapolation of the inputs is compared with its actual value, at the end of the co-simulation step. Iterative approaches such as the ones studied in [23, 25, 324, 325, 329] can readily benefit from this technique.

Parallel Embedded Method: This technique runs a traditional adaptive step size numerical method in parallel with the co-simulation [184]. The purpose is to piggy back in the auxiliary method, the decisions on the step size. The derivatives being integrated in each SU have to be either provided, or estimated.

Conservation Laws: The local error is estimated based on the deviation from a known conservation law. Extra domain knowledge about the coupling between SUs is required. For example, if the couplings form power bonds [294], then energy should be conserved across a co-simulation step. In practice there is always an error due to the usual factors. The magnitude of the energy residual at a start and at end of a co-simulation step serves as an estimate of the local error. This technique has been implemented and studied in [161, 317]. It has the advantage that it may not require rollback functionalities.

Embedded Solver Method: If the individual SUs support adaptive step size, then the decisions made internally can be made public to help the master decide on the communication step size. To the best of our knowledge, there is no master proposed that performs this, but the FMI Standard allows SUs to reject too large communication step sizes [48, 66].

After the error is deemed too large by one of the above methods, the correction can be applied pessimistically (rolling back and repeating the same step) or optimistically (adapt the next step). To mitigate the overhead of a pessimistic approach, the corrections may be applied only to sensitive SUs, as carried out in [372].

Stability

In the previous section we have presented conditions in which a master can reduce the communication step size to an arbitrarily small value in order to meet arbitrary accuracy. Theoretically, this is useful as it tells the master that by reducing the local error, it also reduces the global error. In practice, the communication step size cannot be reduced to an arbitrarily small value without facing performance and round-off error problems. Performance because, for smaller communication step sizes, it takes more steps to compute a behavior trace over a given interval of time. Round-off accuracy because in a digital computer, real numbers can only be represented approximately. Computations involving very small real numbers incur a non-negligible round-off error. So that means that in practice convergence does not imply that arbitrary accuracy can be achieved. A better question is to analyze what happens to the global error, as the co-simulation trace is computed with a non-null communication step size H.

Suppose that the analytical solution to the coupled model induced by the co-simulation scenario eventually goes to zero. This is the case for the coupled multi-body system of

Figure 3.12, described in Equation (3.17), provided that at least one of the constants d_1 or d_2 is positive non-zero. Intuitively, this means that the system will lose energy over time, until it eventually comes to rest.

Let $x_1(t)$ denote the analytical solution of the position of the mass m_1 in the system, and let $\tilde{x_1}(t)$ be the solution computed by a co-SU. Then $e_{x_i}(t) = \|x_1(t) - \tilde{x_1}(t)\|$ denotes the global error at time t made by the co-SU. If $\lim_{t\to\infty} x_1(t) = 0$, then $\lim_{t\to\infty} e_{x_i}(t) = \tilde{x_1}(t)$.

If the co-SU is convergent, then for an arbitrarily small $H \to 0$, $\lim_{t\to\infty} e_{x_i}(t) \to 0$ will be arbitrarily small too. Since in practice we cannot take arbitrarily small H, we want to know whether there is some non-zero H such that $\lim_{t\to\infty} \tilde{x_1}(t) = 0$, thus driving $e_{x_i}(t)$ to zero as well. If that is the case, then it means that, assuming the system will eventually come to rest, the co-SU will too. This property is called numerical stability.

Contrarily to convergence, numerical stability is a property that depends on the characteristics of the system being co-simulated. Numerical stability is always studied assuming that the system being co-simulated is stable. It makes no sense to show that the co-simulation trace will grow unbounded provided that the system does too, as it is a comparison of two infinities.

One of the ways numerical stability in co-simulation can be studied is by calculating the spectral radius (the largest absolute value of the eigen values) of the error in the co-SU, written as an autonomous linear discrete system [72]. This corresponds to the induced model, as introduced in Chapter 2. The following example shows how to compute the induced coupled model, given a master algorithm.

Example 16. Recall that the coupled model being approximated by the co-simulation scenario in Equation (3.25) can be written as:

$$
\begin{aligned}
\begin{bmatrix}\dot{x}_1\\ \dot{v}_1\end{bmatrix} &= \underbrace{\begin{bmatrix}0 & 1\\ -\frac{c_1}{m_1} & -\frac{d_1}{m_1}\end{bmatrix}}_{A_1}\begin{bmatrix}x_1\\ v_1\end{bmatrix} + \underbrace{\begin{bmatrix}0\\ \frac{1}{m_1}\end{bmatrix}}_{B_1} u_1\\
y_1 &= \underbrace{\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}}_{C_1}\begin{bmatrix}x_1\\ v_1\end{bmatrix}\\
\begin{bmatrix}\dot{x}_2\\ \dot{v}_2\end{bmatrix} &= \underbrace{\begin{bmatrix}0 & 1\\ -\frac{c_2+c_c}{m_2} & -\frac{d_c}{m_2}\end{bmatrix}}_{A_2}\begin{bmatrix}x_2\\ v_2\end{bmatrix} + \underbrace{\begin{bmatrix}0 & 0\\ \frac{c_c}{m_2} & \frac{d_c}{m_2}\end{bmatrix}}_{B_2} u_2\\
y_2 &= \underbrace{\begin{bmatrix}c_c & d_c\end{bmatrix}}_{C_2}\begin{bmatrix}x_2\\ v_2\end{bmatrix} + \underbrace{\begin{bmatrix}-c_c & -d_c\end{bmatrix}}_{D_2} u_2
\end{aligned}
\tag{3.42}
$$

with the coupling conditions $u_1 = y_2$ and $u_2 = y_1$.

In order to write the induced co-simulation model as an autonomous linear discrete system, we have to write what happens at a single co-simulation step $t \in [t_n, t_{n+1}]$ when executed by the master presented in Algorithm 3. Since the purpose is to analyze the stability of a co-SU, and not the stability of each of the SUs in the co-simulation, it is common to

assume that the SUs compute the analytical trace of the system. This enables the study of the stability properties of the co-SU, starting from accurate SUs.

From time $t \in [t_n, t_{n+1}]$, SU S_1 is computing the behavior trace of the following Initial Value Problem Ordinary Differential Equation (IVP-ODE):

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{v}_1(t) \end{bmatrix} = A_1 \begin{bmatrix} x_1(t) \\ v_1(t) \end{bmatrix} + B_1 u_1(t_n) \tag{3.43}$$

with initial conditions $\begin{bmatrix} x_1(t_n) & v_1(t_n) \end{bmatrix}^T$ given from the previous co-simulation step. The term $u_1(t_n)$ denotes the fact that we are assuming a constant extrapolation of the input in the interval $t \in [t_n, t_{n+1}]$.

Equation (3.43) is linear and time invariant, so the value of $\begin{bmatrix} x_1(t_{n+1}) \\ v_1(t_{n+1}) \end{bmatrix}$ can be given analytically as:

$$\begin{bmatrix} x_1(t_{n+1}) \\ v_1(t_{n+1}) \end{bmatrix} = e^{A_1 H} \begin{bmatrix} x_1(t_n) \\ v_1(t_n) \end{bmatrix} + \left(\int_{t_n}^{t_{n+1}} e^{A_1 (t_{n+1} - \tau)} d\tau \right) B_1 u_1(t_n) \tag{3.44}$$

or, replacing the integration variable with $s = \tau - t_n$,

$$\begin{bmatrix} x_1(t_{n+1}) \\ v_1(t_{n+1}) \end{bmatrix} = e^{A_1 H} \begin{bmatrix} x_1(t_n) \\ v_1(t_n) \end{bmatrix} + \underbrace{\left(\int_0^{H} e^{A_1 (H - s)} ds \right)}_{K_1} B_1 u_1(t_n) \tag{3.45}$$

where $e^X = \sum_{k=0}^{\infty} \frac{1}{k!} X^k$ is the matrix exponential.

Rewriting Equation (3.45) as a discrete time system gives us the computation performed by SU S_1 in a single co-simulation step, that is, the state transition function δ_1:

$$\begin{bmatrix} x_1^{(n+1)} \\ v_1^{(n+1)} \end{bmatrix} = e^{A_1 H} \begin{bmatrix} x_1^{(n)} \\ v_1^{(n)} \end{bmatrix} + K_1 B_1 u_1^{(n)} \tag{3.46}$$

where $z^{(n)} = z(t_n)$.

At the end of the co-simulation step ($t = t_{n+1}$) the output of the first SU, that is, its output function λ_1, is given by plugging in Equation (3.46) to the output y_1 in Equation (3.42):

$$y_1^{(n+1)} = C_1 e^{A_1 H} \begin{bmatrix} x_1^{(n)} \\ v_1^{(n)} \end{bmatrix} + C_1 K_1 B_1 u_1^{(n)} \tag{3.47}$$

Repeating the same procedure for the second SU, yields the state transition δ_2 and output functions λ_2:

$$\begin{aligned} \begin{bmatrix} x_2^{(n+1)} \\ v_2^{(n+1)} \end{bmatrix} &= e^{A_2 H} \begin{bmatrix} x_2^{(n)} \\ v_2^{(n)} \end{bmatrix} + K_2 B_2 u_2^{(n)} \\ y_2^{(n+1)} &= C_2 e^{A_2 H} \begin{bmatrix} x_2^{(n)} \\ v_2^{(n)} \end{bmatrix} + \left(C_2 K_2 B_2 + D_2 \right) u_2^{(n)} \end{aligned} \tag{3.48}$$

with $K_2 = \int_0^H e^{A_2(H-u)}du$.

Since the coupling conditions are $u_1 = y_2$ and $u_2 = y_1$, we can combine Equations 3.48, 3.47, and 3.43 into a single discrete time system:

$$\begin{bmatrix} \begin{bmatrix} x_1^{(n+1)} \\ v_1^{(n+1)} \end{bmatrix} \\ y_1^{(n+1)} \\ \begin{bmatrix} x_2^{(n+1)} \\ v_2^{(n+1)} \end{bmatrix} \\ y_2^{(n+1)} \end{bmatrix} = \underbrace{\begin{bmatrix} e^{A_1 H} & \bar{0} & \bar{0} & K_1 B_1 \\ C_1 e^{A_1 H} & \bar{0} & \bar{0} & C_1 K_1 B_1 \\ \bar{0} & K_2 B_2 & e^{A_2 H} & \bar{0} \\ \bar{0} & C_2 K_2 B_2 + D_2 & C_2 e^{A_2 H} & \bar{0} \end{bmatrix}}_{A} \begin{bmatrix} \begin{bmatrix} x_1^{(n)} \\ v_1^{(n)} \end{bmatrix} \\ y_1^{(n)} \\ \begin{bmatrix} x_2^{(n)} \\ v_2^{(n)} \end{bmatrix} \\ y_2^{(n)} \end{bmatrix} \tag{3.49}$$

The above system represents the model induced by the co-simulation algorithm. It is stable if the behavior traces remain bounded (e.g., by going to zero) as $n \to \infty$. This can be checked by observing whether the spectral radius $\rho(A) < 1$. For parameters $m_1 = m_2 = c_1 = c_2 = d_1 = c_c = d_c = 1, d_2 = 2$, a communication step size of $H = 0.001$, $\rho(A) = 0.9992$, which means that the co-SU is stable. If the damping constant were $d_c = 6.0E6$, then the co-SU would be unstable ($\rho(A) \approx 76.43$). A stable co-simulation is shown in Figure 3.17.

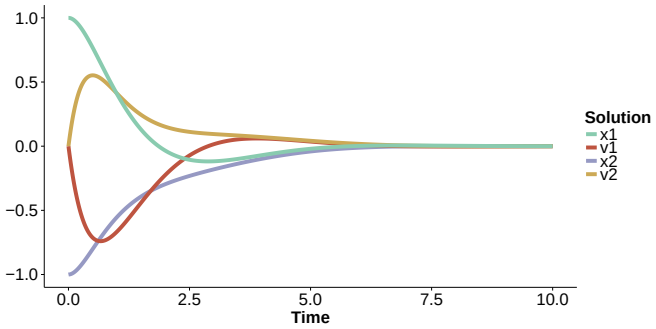


Figure 3.17: Behavior trace of co-simulator described in Equation (3.49). Parameters are: $m_1 = m_2 = c_1 = c_2 = d_1 = c_c = d_c = 1, d_2 = 2, H = 0.001$.

As Example 16 shows, the numerical stability is determined by the internal solver of each SU, by the input approximation scheme, the master algorithm, and the communication step size.

In Chapters 5 and 6, we revisit stability in more detail. Additionally, see [71, 72, 74] for the stability analysis under multiple coupling approaches and approximating functions. Stability of various co-SUs has been also studied in [23, 165, 201, 216, 325]. The *rules of thumb* drawn from these papers can be summarized as: (1) Co-simulators that employ fixed point iteration techniques typically have better stability properties; (2) Gauss-Seidel coupling approach has slightly better stability properties when the order in which the SUs compute is appropriate (e.g., the SU with the highest mass should be computed first [23]).

In industrial problems, the stability analysis of a co-simulation algorithm is seldom done. This is because the SUs are black boxes (to protect IP), so there is little knowledge about the kind of solver and model being used and its stability properties. In these situations, we recommend that strong coupling algorithms be used, as these have better stability properties. However, these techniques require rollback functionalities which can be difficult to support for certain SUs. Even if those functionalities are available, the cost of computing a co-simulation trace can be prohibitively high when compared with non-iterative approaches.

Compositional Continuity

If a SU approximates the behavior of a CT system, then it is reasonable to expect that its inputs are also continuous. As discussed in [71, 318], the careless use of input extrapolations (e.g., constant extrapolation) may violate this assumption. Consider the point of view of a SU S_i in co-simulation. Throughout a co-simulation step $t \in [t_n, t_{n+1}]$ the input $\phi_{u_i}(t, u_i(t_n)) = u_i(t_n)$ is kept constant. At the next co-simulation step $t \in [t_{n+1}, (n+2)H]$, the input $\phi_{u_i}(t, u_i(t_{n+1})) = u_i(t_{n+1})$ may change radically if $u_i(t_{n+1})$ is too far away from $u_i(t_n)$.

Any *sudden* change in the input to a CT SU may wreak havoc in the performance of its simulator, causing it to reduce inappropriately the internal micro step size, to reinitialize the solver [87, 373], to discard useful information about the past (in multi-step solvers [19, 20]), and/or produce inaccurate values in its input extrapolation [285]. Furthermore, a discontinuity may be propagated to other SUs, aggravating the problem.

We now discuss the continuity problem for simulation, and later relate it to co-simulation.

Most numerical methods assume that the input is a discretized version of a continuous trace. That means that, when a discontinuity occurs, SU S_i cannot distinguish it from a very steep change in the continuous trace. The way traditional solvers deal with this behavior is to reduce the micro step size h_i until the change is not so steep. This works with a continuous signal with a steep change, but does not work with a discontinuity: even if the micro-step size h_i is reduced, the difference between $\lim_{t\to(t_{n+1})^-} \phi_{u_i}(t, u_i(t_n)) = u_i(t_n)$ and $\lim_{t\to(t_{n+1})^+} \phi_{u_i}(t, u_i(t_{n+1})) = u_i(t_{n+1})$ is still the same, as it depends on the communication step size H and not on the micro step size h_i. The solver will reduce the micro step size until a minimum is reached, at which point it gives up and finally advantages the micro step [87].

Most of the times this gives acceptable results but has a huge performance toll: when the solver is repeatedly retrying a small micro-step size, it does not advance the simulated time. This means that a huge computational effort goes to waste until the solver finally gives up [85].

We defer the discussion of the correct ways to deal with discontinuities to co-simulation scenario where discontinuities are expected, Section 3.8. In continuous co-simulation scenarios, discontinuities should not occur.

A solution to avoid discontinuities in the input approximations is to use the extrapolated interpolation methods [71, 116], exemplified below. These methods ensure at least that the left and right limit of the exchanged data points match: $\lim_{t\to(t_{n+1})^-} \phi_{u_i}(t, u_i(t_n)) = \lim_{t\to(t_{n+1})^+} \phi_{u_i}(t, u_i(t_{n+1}))$.

Example 17. To give an example, we derive one possible linear extrapolated interpolation method for ϕ_{u_i} over the interval $t \in [t_n, t_{n+1}]$. Since ϕ_{u_i} is linear, then

$$\phi_{u_i}(t, u_i(t_n), u_i(t_{n-1})) = b + a(t - t_n)$$

, for some constants a, b. Let $\bar{u}_i(t_n) = \phi_{u_i}(t_n, u_i(t_{n-1}), u_i((n-2)H))$. To avoid discontinuities, we require that $\phi_{u_i}(t_n, u_i(t_n), u_i(t_{n-1})) = \bar{u}_i(t_n)$. And we want that $\phi_{u_i}(t_{n+1}, u_i(t_n), u_i(t_{n-1})) = u_i(t_n)$.

So putting these constraints together gives

$$\begin{aligned} \phi_{u_i}(t, u_i(t_n), u_i(t_{n-1})) &= b + a(t - t_n) \\ \bar{u}_i(t_n) &= \phi_{u_i}(t_n, u_i(t_{n-1}), u_i((n-2)H)) \\ \phi_{u_i}(t_n, u_i(t_n), u_i(t_{n-1})) &= \bar{u}_i(t_n) \\ \phi_{u_i}(t_{n+1}, u_i(t_n), u_i(t_{n-1})) &= u_i(t_n) \end{aligned} \tag{3.50}$$

Solving this system for $\phi_{u_i}(t, u_i(t_n), u_i(t_{n-1}))$ gives:

$$\phi_{u_i}(t, u_i(t_n), u_i(t_{n-1})) = u_i(t_{n-1}) + \frac{u_i(t_n) - u_i(t_{n-1})}{H}(t - t_n) \tag{3.51}$$

Real-time Constraints, Noise, and Delay

The major challenge in real-time co-simulation is to ensure that a SU is fast-enough to satisfy the timing constraint $t = \alpha\tau$. In real-time co-simulation, this challenge gets aggravated due to the presence of multiple SUs, with different capabilities [340], and whose internal details are unknown. Furthermore, real-time co-simulation is often used when at least one of the SUs is a physical entity (as in Section 3.2.2). This means that measurements may carry noise, and the extrapolation functions used in the other SUs have to be properly protected from that noise. Finally, the quality of the network is important, as the real-time SUs needs to receive their inputs in a timely manner. To mitigate this, the master algorithm has to compensate for any delays in the receiving of data, and provide inputs to the real-time SU [339].

3.7.4 Standards for CT Co-simulation

In the realm of continuous time based standards, we highlight the Dynamical System Block (DSBlock) [287] standard (early 90s), whose purpose was not strictly to enable co-simulation, but to be able to represent differential equation based models in a uniform way. Then numerical solvers could be used to simulate these models. This standard later inspired the creation of the Functional Mockup Interface (FMI) standard [47] (late 2000s) for co-simulation. Contrarily to discrete event based standards, continuous based standards do not attempt to standardize the interaction between simulators. This is because there is no single best way to coordinate a continuous time co-simulation, and often extra knowledge and experience are required to pick the best way.

The above concepts try to capture the essence of continuous time co-simulation. Note that a CT SU does not have to be a mockup of a CT system. We introduce them as such to simplify the explanation. Most common co-simulation scenarios will include SUs that share continuous and discrete event characteristics. In the next section, we focus on these.

3.8 Hybrid Co-simulation Approach

Sections 3.6 and 3.7 described the essential characteristics and assumptions of simulation units (SUs) for each kind of co-simulation approach. When compared to a CT SU, whose state evolves continuously in time and whose output may have to obey physical laws of continuity, a DE SU state can assume multiple values at the same time (transiency) and its output is discontinuous. For a master algorithm, a CT SU has some flexibility (except for algebraic loops and complex coupling conditions) in deciding the parameters (e.g., step size or tolerance) of the co-simulation. In contrast, a DE SU has to get inputs and produce outputs at the precise time an event is supposed to occur, and there is no Lipschitz continuity conditions to help predict how a delay in the output of the DE SU can affect the overall co-simulation trace.

These differences are at the heart of many challenges in hybrid co-simulation scenarios, which are studied in detail in Chapter 6.

Please refer to [260, 262, 270, 271] for examples of hybrid systems, and descriptions of their characteristics.

3.8.1 Hybrid Co-simulation Scenarios

We do not give a formal definition of a hybrid co-simulation scenarios because that is related to finding an appropriate standard for hybrid co-simulation and master algorithms that would solve the challenges enumerated in Section 3.8.2.

Instead, we define it broadly as mixing the characteristics and assumptions of both kinds of SUs. These scenarios, together with an adequate master, can be used as mock-ups of hybrid systems [17, 80, 84, 246].

Example 18. A thermostat regulating the temperature in a room is a classical example [242]. The Continuous Time (CT) constituent system represents the temperature dynamics of the room, accounting for a source of heat (radiator). The Discrete Event (DE) part is a controller that turns on/off the radiator depending on the temperature.

The SU S_1 simulates the following dynamics:

$$\dot{x} = -\alpha\,(x - 30q)\,; \qquad x(0) = x_0 \tag{3.52}$$

where x is the output temperature in the room, $\alpha > 0$ denotes how fast the room can be heated (or cooled) down, and $q \in \{0, 1\}$ is the control input that turns on/off the radiator. The SU S_2 simulates the statemachine shown in Figure 3.18, where one can think of the input event $tooHot$ as happening when $x(t) \geq 21$ and $tooCold$ when $x(t) \leq 19$. The output events *off* and on will assign the appropriate value to the input q of S_1. Therefore, the temperature $x(t)$ is kept within a comfort region.

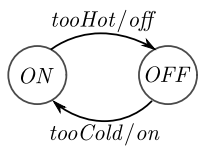


Figure 3.18: Statemachine model of the controller constituent system.

The two SUs in Example 18 cannot just be coupled together via input to output assignments. Any master for this co-simulation scenario has to reconcile the different assumptions about the inputs and output of each SU.

- The CT SU expects a continuous input, whereas the output of the DE SU is an event signal.
- The output of the CT SU is a continuous signal, whereas the DE SUs expects an event signal as input.

The coupling of CT and DE black box SUs has been studied in the state of the art. In essence, two approaches are known, both based on adapting (or wrapping) the behavior of the SU:

Hybrid DE – adapt every CT SU as a DE SU, and use a DE based master;
Hybrid CT – wrap every DE SU to become a CT SU and use a CT based master.

We will denote these techniques as semantic adaptations, defined in Chapter 7.

According to the formalization that we have proposed for CT and DE SUs, the *Hybrid DE* approach, applied to the thermostat example involves:

1. adapting S_1 as a DE SU, S_1', with a time advance that matches the size of the co-simulation step;
2. keeping track of the output of S_1 in order to produce an output event whenever it crosses the thresholds; and, conversely,
3. any output event from S_2 has to be converted into a continuous signal for the input $q(t)$ of S_1.

Other examples of *Hybrid DE* are described in [31, 53, 54, 78, 79, 125, 211, 213, 220, 274, 282, 284, 310, 366, 376, 382, 384].

To apply the *Hybrid CT* to Example 18, we require the adaptation of the DE S_2 as a CT SU that takes as input the temperature continuous signal, and internally reacts to an event caused by the crossing of the threshold. The output event of S_2 can be converted into a continuous signal $q(t)$. Examples of the *Hybrid CT* include [110, 122, 139, 227, 309, 345, 353].

Regardless of the approach taken, care must be taken to uphold the properties of the coupled model: the fact that an otherwise discontinuous signal becomes continuous as a result of a linear or higher order extrapolation may violate these properties. Knowledge of the domain and the SUs is paramount to retain aforementioned properties.

In the section below, different challenges that arise in hybrid co-simulation will be discussed.

3.8.2 Challenges

Semantic Adaptation and Model Composition

While a generic wrapper based on the underlying model of computation of the SU can be used, as done in [100, 308], the realization of any of the approaches *Hybrid DE* or *Hybrid CT* depends on the concrete co-simulation scenario and the features of the SUs [61, 266], as shown with the thermostat example. As Chapters 5 and 7 show, there is no best choice of wrappers for all scenarios. Even at the technical level, the manner in which the events

or signals are sent to (or obtained from) the SU may need to be adapted [156, 353]. To account for this variability, the most common adaptations can be captured in a configuration language, as was done in [110, 251] and Chapter 7, or in a specialization of a model of computation, as done in [220, 264, 296]. These approaches require that a person with the domain knowledge describes how the SUs can be adapted.

Our choice of wrapper for the *Hybrid DE* approach is meant to highlight another problem with the adaptations of SUs: the wrapper incorporates information that will ultimately have to be encoded in the software controller. As such, we argue that the need for sophisticated semantic adaptations should be smaller in later stages of the development of the components so that, for more refined models of the thermostat, the decision about when to turn off the radiator is not made by a wrapper of S_1.

Predictive Step Sizes and Event Location

In the *Hybrid DE* approach, the time advance has to be defined (recall Equation (3.2)). Setting it to whatever co-simulation step size H the master decides, will work, but the adapted SU may produce many absent output events. Better adaptations have been proposed. In the thermostat example, S_1' can propose a time advance that coincides with the moment that $x(t)$ will leave the comfort region, thereby always being simulated at the relevant times. Naturally, these approaches rely on information that may expose the IP of SUs.

Others try to adaptively guess the right time advance by monitoring other conditions of interest, set over the own dynamics of the adapted SU, the most common approach being the quantization of the output space [54, 210, 211, 283, 385].

The capability to predict the time advance is also useful to enhance the performance/accuracy of CT based co-simulation, as shown in [66].

Locating the exact time at which a continuous signal crosses a threshold is a well known problem [55, 57, 388] and intimately related to guessing the right time advance for predicting the step size [79, 136]. To address this, solutions typically require derivative information of the signal that causes the event, and/or the capability to perform rollbacks.

In Example 18, a co-simulation that shows the output q of the controller changing from 0 to 1 at time t_e while the temperature of the room x actually crossed the comfort zone at $t_e - k$, for $k > 0$, may not be accurate if k is too large. Note that k is a consequence of the decisions made in the master.

Discontinuity Identification

Until here, we have based our discussion in the knowledge of what kind of SUs comprise a co-simulation. In the most common implementations of co-simulation, a signal is often represented as a set of time-stamped points. Observing this sequence of points alone does not make it possible to discern a steep change in a continuous signal, from a true discontinuity, that occurs in an event signal [67, 232, 260, 388]. Extra information is currently used: *a*) a formalization of time which include the notion of absent signal, as proposed in [67, 232, 345]; or *b*) an extra signal can be used to discern when a discontinuity occurs, as done in the FMI for Model Exchange [48], even facilitating the location of the exact time of the discontinuity; or *c*) symbolic information (e.g., Dirac impulses [112]) that characterize a discontinuity can be included, as done in [160, 278].

Discontinuity Handling

Once a discontinuity is located, how it is handled depends on the nature of the SUs and their capabilities. If the SU is a mock-up of a continuous system then, traditionally, discontinuities in the inputs should be handled by reinitializing the SU [87]. This step can incur a too high performance cost, especially with multi-step numerical methods, and [19, 20] proposes an improvement for these solvers.

A discontinuity can cause other discontinuities at the same simulated time, triggering a cascade of re-initializations. During this process, which may not finish, care must be taken to ensure that physically meaningful properties such as energy distribution, are respected [261].

Algebraic Loops, Legitimacy, and Zeno Behavior

Algebraic loops are non-causal dependencies between SUs that can be detected using feedthrough information, as explained in Section 3.7.3. In CT based co-simulation, the solution to algebraic loops can be attained by a fixed point iteration technique, as covered in Section 3.7.3. There is the possibility that the solution to an algebraic loop will fail to converge.

In DE based co-simulation a related property is legitimacy [386], which is roughly the undesirable version of the *transiency* property, explained in Section 3.6. An illegitimate co-simulation scenario will cause the co-simulation master to move an infinite number of events with the same timestamp between SUs, never advancing time. Distance matrices, used to optimize parallel optimistic approaches, as explained in [135] and used in [144], can be leveraged to detect statically the presence of *some* classes of illegitimacy.

A similar behavior, but more difficult to detect is Zeno behavior. It occurs when there is successively smaller intervals of time between two consecutive events, up to the point that the sum of all these intervals is finite [360]. As shown in [65], a simulator eventually fails to detect the consecutive events. In particular, it advocates that the Zeno behavior is a property of the model, whereas the incorrectness is due to a simulation approximation error. However, while illegitimate behaviors are not desired in pure DE co-simulation, Zenoness can be a desired feature in some hybrid co-simulation scenarios (e.g., see [63]). We say in the theoretical sense because, for the purposes of co-simulation, scenarios with Zenoness still have to be recognized and appropriate measures, such as regularization [194], have to be taken.

Stability under X

If a hybrid co-simulation represents a hybrid or switched system [360], then it is possible that a particular sequence of events causes the system to become unstable, even if all the individual continuous modes of operation are stable [197, Example 1.1]. New analyses are required to identify whether the CT SUs can yield unstable trajectories as a result of: 1. noisy inputs; 2. data quantization; 3. change of co-simulation orchestration [155]; 4. the events of wrapped DE SUs [153]; and, 5. delayed exchange of values.

Example analyses are developed in Chapters 5 and 6.

Theory of DE Approximated States

In a pure DE based co-simulation, if round-off errors are neglected, the computed trajectories are essentially exact. To the best of our knowledge, only [386] addresses theoretically how the error in a discrete event system can be propagated. In CT based co-simulation however, error is an accepted and well studied and techniques exist to control it.

In Hybrid co-simulation, there is a need for analysis techniques that provide bounds on the error propagation in the DE SUs, when these are coupled to sources of error. The reason why this is so difficult is discussed in Chapter 6.

In addition, based on these analyzes, it should be possible for a DE SU to recognize that its error has exceeded a given tolerance, and measures should be taken to reduce that error. Having these techniques in place allows a hybrid co-simulation master to take appropriate measures (e.g., adapt the communication step size, etc...) the keep the error bounded in every SU.

3.8.3 Standards for Hybrid Co-simulation

In the hybrid co-simulation domain, there have been some efforts to standardize both the orchestration and interfaces.

While for CT co-simulation there is the Functional Mockup Interface (FMI) standard [48], and for DE co-simulation there is the High Level Architecture (HLA) [14] standard, as of the time of writing, both standards have limitations for hybrid co-simulation. References [52, 99, 139, 345] use/propose extensions to the FMI standard and [30] proposes techniques to perform CT simulation conforming to HLA. Recognizing that hybrid co-simulation is far from well studied, [67] proposes a set of idealized test cases that any hybrid co-SU, and underlying standard, should pass. In particular, it is important to have correct handling and representation of time, to achieve a sound approach for simultaneity.

While there is no formal standard, there are plenty of potential candidates. We highlight[6]: the Ptolemy project [68], where a key principle is the use of multiple models of computation in a hierarchical heterogeneous design environment; ModHel'X [61], which makes those models of computation customizable by the modeler; DEVS&DESS [384], which builds on DEVS to allow the representation of continuous behavior; and HFSS [34], which decouples the transmission of data between simulators, and allows for the simulation of dynamic structure systems.

Finally, even with a standardized interface, SUs have different capabilities: a fact that makes coding an optimal master algorithm difficult. A possible approach to deal with this heterogeneity, proposed in [150, 156] and Chapter 7, is to assume that all SUs implement the same set of features, code the master algorithm for those features, and delegate to wrappers the responsibility of leveraging extra features (or mitigating the lack of). In the section below, these features are classified.

[6]The field in hybrid systems is vast and here we restrict our scope to standards that focus on the simulation of such systems (e.g., we do not consider Hybrid Automata [243] or Hybrid Programs [302] as candidates because their primal intent is analysis of hybrid systems).

3.9 Classification and Applications

Having described the multiple facets of co-simulation, this section summarizes our classification and methodology, and applies it to a typical use case.

3.9.1 Methodology

To find an initial set of papers related to co-simulation, we used Google Scholar with the keywords "co-simulation", "cosimulation", "coupled simulation", and collected the first 10 pages of papers. Every paper was then filtered by the abstract, read in detail, and its references collected. To guide our reading to the most influential papers, we gave higher priority to most cited (from the papers that we have collected).

We read approximately 30 papers to create the initial version of the taxonomy. Then, as we read new papers, we revised the taxonomy and classified them.

After a few iterations, new references did not cause revisions to the taxonomy, which prompted us to classify the collected papers in a more systematic fashion: all the papers that we collected from 2011 (inclusive) up to, and including, 2016 were classified. Two main reasons justify the last 5 years interval: limited time; and most of the papers refer to, and are based on, prior work. As a consequence, the classification would be very similar for many of the related references prior to 2011.

In total, 84 papers were classified.

3.9.2 Taxonomy

The taxonomy is represented as a feature model [202] structured in three main categories, shown in Figure 3.19:

Non-Functional Requirements (NFRs): Groups concerns (e.g., performance, accuracy, and IP Protection) that the reference addresses.

SU Requirements (SRs): Features required/assumed from the SUs by the master described in the paper. Examples: Information exposed, causality, local/remote availability, or rollback support.

Framework Requirements (FRs): Features provided by the master. Examples: dynamic structure, adaptive communication step size, or strong coupling support.

Each main group is detailed in Figures 3.20 to 3.22. Abstract features denote concepts that can be easily detailed down but we chose not to, for the sake of brevity. Mandatory features are required for the activity of co-simulation, while optional are not.

3.9.3 Applications

To demonstrate how the taxonomy is used, we picked three representative examples from the state of the art: an industrial use case, a co-simulation framework, and a co-simulation standard.

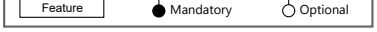


Figure 3.19: Top-level.

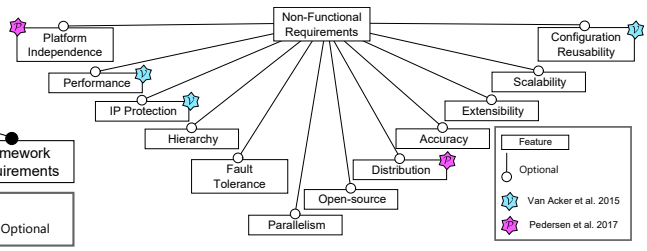


Figure 3.20: Non-Functional Requirements.

3.9.3.1 An Industrial Application

The case study presented in Section 3.3.1 is a representative example application because: it includes parts that are developed by other departments (e.g., the ship engine) and external suppliers (e.g., the water treatment system); there are both continuous and discrete event dynamics (e.g., the control system is comprised of a state machine and a PI-Controller); and, quoting the authors, "to improve the control strategy of the WHS, a higher-fidelity model [of the systems interacting with the controller] should be used." [297, Section 3.4].

This work is classified as highlighted in Figures 3.20 to 3.22.

3.9.3.2 A Framework

We next consider the work of [358], where an FMI based multi-rate master algorithm is generated from a description of the co-simulation scenario. In the paper, the description language introduced can be reused in a tool-agnostic manner. The orchestration code generator analyzes the co-simulation scenario, and: a) identifies algebraic loops using I/O feed-through information; b) separates the fast moving SUs from the slow moving ones, using the preferred step size information, and provides interpolation to the fast ones (multi-rate); and c) finds the largest communication step size that divides all step sizes suggested by SUs and uses it throughout the whole co-simulation. The algebraic loops are solved via successive substitution of inputs, storing and restoring the state of the SUs.

Based on these facts, [358] is classified as highlighted in Figures 3.20 to 3.22.

3.9.3.3 A Standard

The FMI standard for co-simulation, version 2.0 [48, 126], defines the interface and interaction pattern that allows simulation units to communicate.

It can be considered a restricted version of the formalization proposed in Section 3.7, with the following differences:

- Each simulation unit is distributed as a zip file, containing binary libraries that can be loaded in compatible execution architectures;
- These binaries implement a standardized software interface that allows the master to set/get inputs, and execute a co-simulation step;
- There is an explicit initialization mode, where a fixed point iteration can be run to find a consistent set of initial values;
- The simulation unit can optionally expose their internal state;

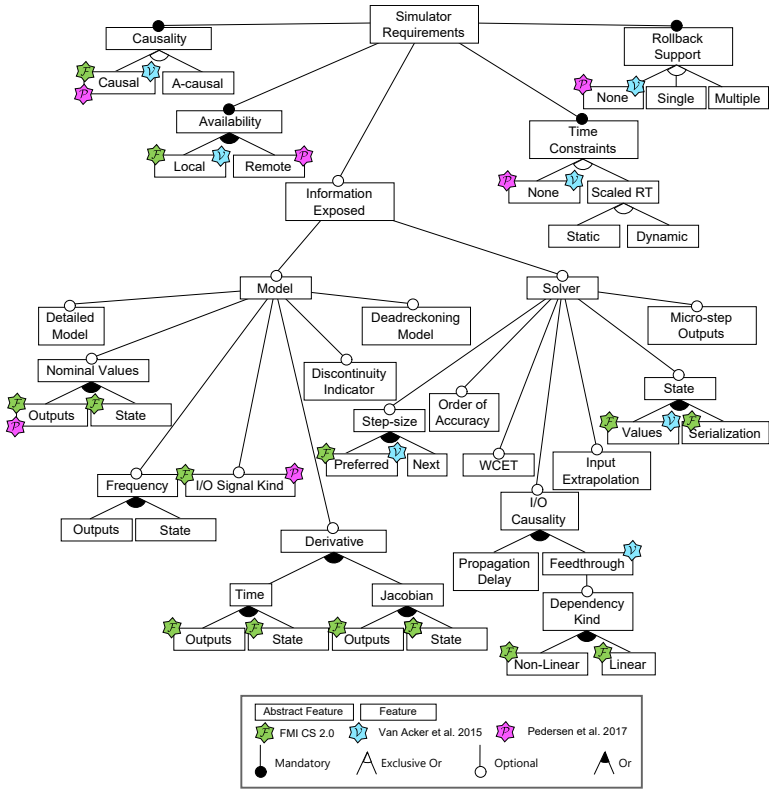


Figure 3.21: Simulation Unit Requirements and features provided in the FMI Standard for co-simulation, version 2.0.

- Outside the initialization mode, it is *not* possible to perform a fixed point iteration on the output variables only (a workaround is to use a strong coupling technique); and
- the output function depends only on the internal state (see [126, p. 104] and [348] for consequences).

Taking these differences into account, the standard can be classified according to the assumptions it makes about the participating SUs. This is highlighted in Figure 3.21.

3.9.4 The State of the Art

The remaining state of the art is classified in Figs. 3.23–3.26. The raw data is available online[7]. The apparent lack of papers in the interval 2006-2009 is a consequence of our methodology (recall Section 3.9.1).

[7] `http://msdl.cs.mcgill.ca/people/claudio/pub/Gomes2016bClassificationRawData/raw_data.zip`

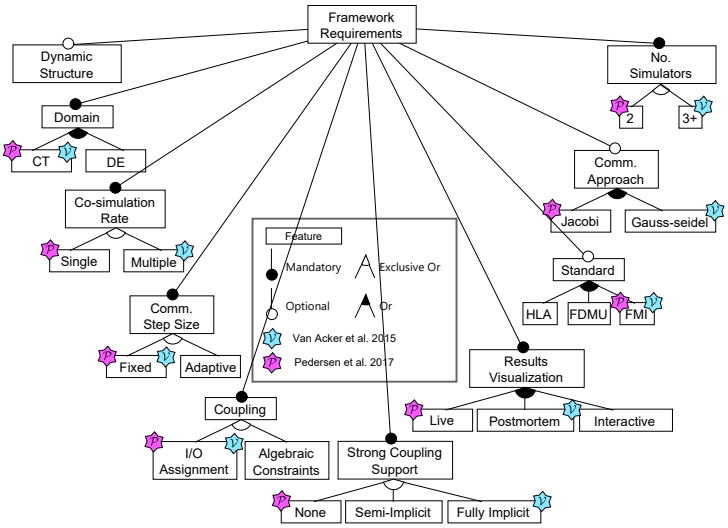


Figure 3.22: Framework Requirements.

3.9.5 Discussion

Analyzing Figure 3.23, Accuracy is the most observed NFR, with 31 reports, followed by IP protection and Performance. This is consistent with the grand challenge in co-simulation (Section 3.2).

The least observed NFRs are Fault tolerance, Hierarchy and Extensibility. Fault tolerance is especially important for long running co-simulations. One of the industrial partners of the INTO-CPS project has running co-simulations that takes a minimum of two weeks to complete.

We argue that Extensibility (the ability to easily accommodate new features) should be given more importance: if an heterogeneous set of SUs participate in the same co-simulation scenario, the combination of capabilities provided (see Figure 3.21) can be huge. Thus, the master can either assume a common homogeneous set of capabilities, which is the most common approach, or can leverage the capabilities provided by each one. In any case, extensibility and hierarchy are crucial to address, and implement, new semantic adaptations.

As Figure 3.25 suggests, we could not find approaches that make use of the nominal values of state and output variables, even though these are capabilities supported in the FMI Standard, and are useful to detect invalid co-simulations. A-causal approaches are important for modularity, as explained in Section 3.7.3, but these are scarce too.

As for the framework requirements, in Figure 3.26, the least observed features are dynamic structure co-simulation, interactive visualization, multi-rate, algebraic coupling, and partial/full strong coupling support. This can be explained by the fact that these features depend upon the capabilities of the SUs, which may not be mature.

Figs. 3.23 – 3.26 do not tell the full story because they isolate each feature. Feature interaction is a common phenomenon, and among many possible interactions, we highlight

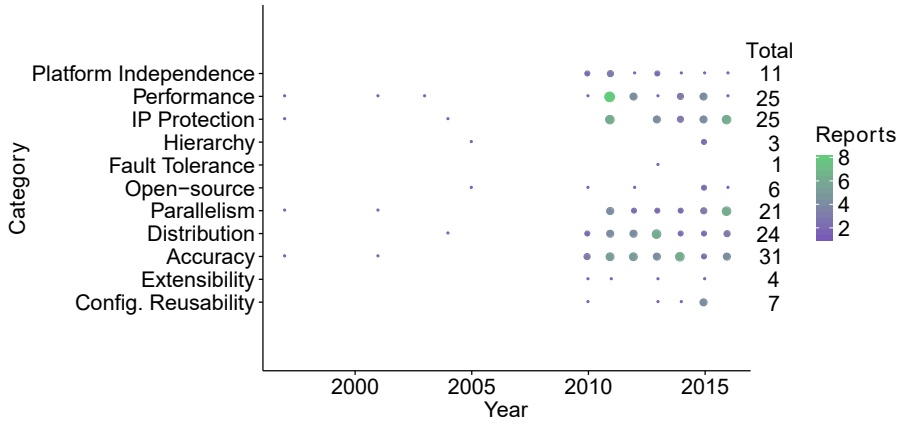


Figure 3.23: Classification with respect to non-functional requirements.

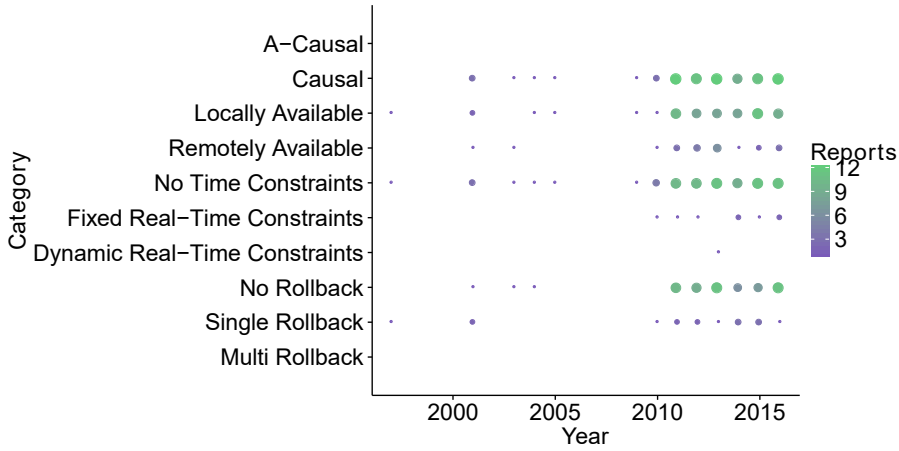


Figure 3.24: Classification with respect to SU requirements: execution capabilities.

the accuracy concern, domain of the co-simulation, number of SUs supported, and IP protection. As can be seen from Figure 3.28, there is only one approach [220] that is both CT and DE based, up to any number of SUs. Accommodating the different CT and DE domains means that the approach assumes that the SUs can behave both as a CT and as a DE SU.

The concern with IP protection is evident in Figure 3.23 but the number of DE and CT based approaches that provide some support for it is small, as shown in Figure 3.27. Similarly, as Figure 3.29 suggests, accuracy does not show up a lot in the DE and CT approaches, for more than two SUs. Accuracy is particularly important in interactions between DE and CT SUs.

In general, from the observed classification, there is a lack of research into approaches that are both DE and CT based, and that leverage the extra features from the SUs.

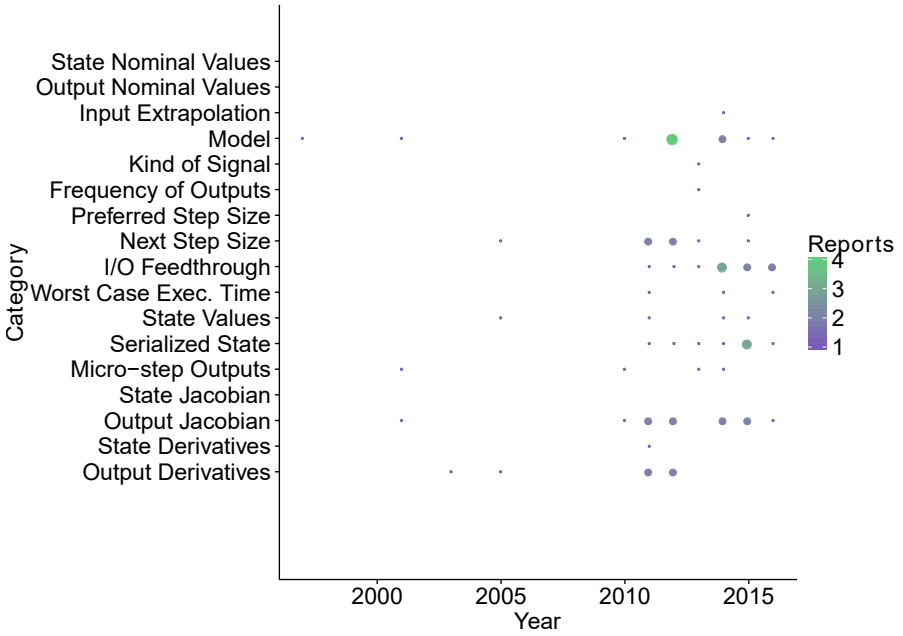


Figure 3.25: Classification with respect to SU requirements: information exposed.

3.10 Concluding Remarks

In this overview chapter, we show that there are many interesting challenges to be explored in co-simulation, which will play a key role in enabling the virtual development of complex heterogeneous systems in the decades to come. The early success can be attributed to a large number of reported applications. However, the large majority of these applications represent *ad-hoc* couplings between two simulators of two different domains (e.g., a network simulator with a power grid one, or a HVAC simulator with a building envelop one)[8]. As systems become increasingly complex, the demand for co-simulation scenarios that are large, hierarchical, heterogeneous, accurate, IP protected, and so on, will increase.

This survey covers the main challenges in enabling co-simulation. To tackle such a broad topic, we have covered two main domains—continuous-time- and discrete-event-based co-simulation—separately and then discussed the challenges that arise when the two domains are combined. A taxonomy is proposed and a classification of the works related to co-simulation in the last five years is carried out using that taxonomy.

From the challenges we highlight: semantic adaptation, modular coupling, stability and accuracy, finding a standard for hybrid co-simulation, and configuration of master algorithms that leverage the individual capabilities of simulation units. For early system analysis, the adaptations required to combine simulators from different formalisms, even conforming to the same standard, are very difficult to generalize to any co-simulation scenario.

One of the main conclusions of the classification is that there is lack of research into modular, stable, and accurate coupling of simulators in dynamic structure scenarios. This is

[8] We did not consider the (potentially many) unreported applications of co-simulation.

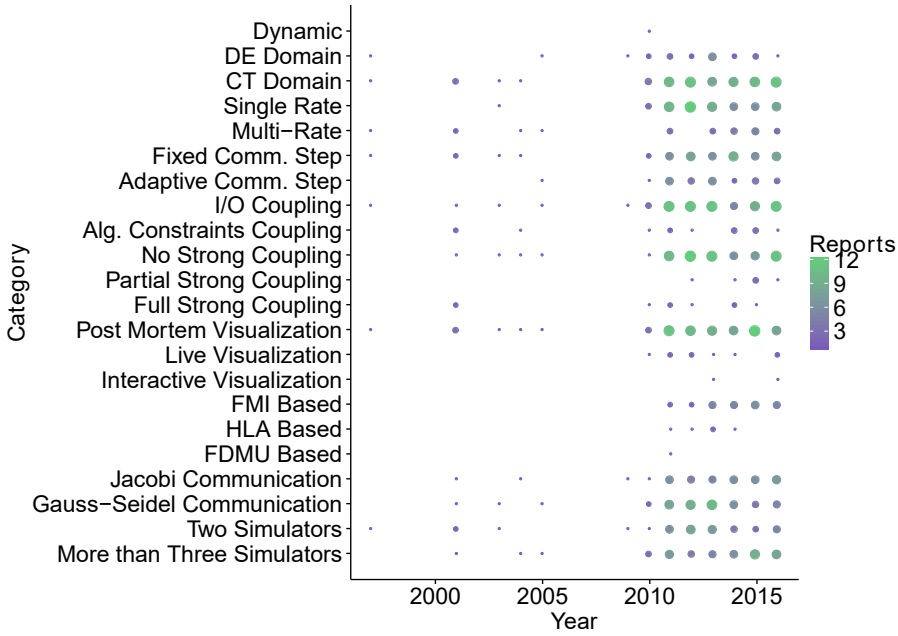


Figure 3.26: Classification with respect to framework requirements.

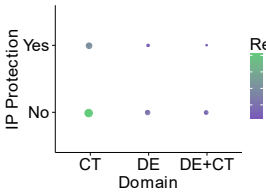


Figure 3.27: Formalisms vs IP Protection.

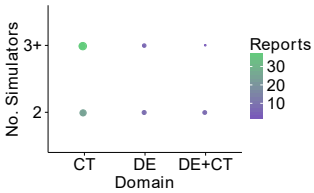


Figure 3.28: Formalisms vs SUs.

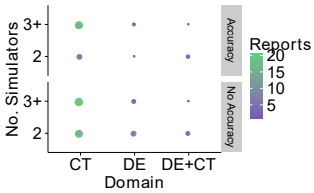


Figure 3.29: Accuracy vs Formalisms vs SUs.

where a-causal approaches for co-simulation can play a key role because they allow the same simulator to be coupled in many different ways. The use of bi-directional effort/flow ports can be a solution inspired by Bond-graphs [294], and there is some work already in this direction [161, 318].

Limitations. The sources of information for the challenges identified in this chapter where existing published works, and discussions with the co-authors. We did not contact other experts and practitioners in the field. As such, it is difficult to priority these challenges. The next chapter attempts to tackle this, by reporting on an empirical survey among experts and practitioners of co-simulation.

Chapter 4

Empirical Survey

Disclaimer The content in this chapter is adapted from:

- Schweiger, Gerald, Cláudio Gomes, Georg Engel, Irene Hafner, Josef Schoeggl, Alfred Posch, and Thierry Nouidui. "Functional Mock-up Interface: An Empirical Survey Identifies Research Challenges and Current Barriers." In The American Modelica Conference. Cambridge, MA, USA, 2018.
- Ongoing work on a manuscript submitted to the Simulation Modelling Practice and Theory journal.

The challenges identified in the previous chapter lack input from experts in industry.

This work complements the existing surveys by providing the empirical aspect. We interviewed multiple experts from various fields in industry and academia as part of a two-stage Delphi study. As a result, the current challenges, research needs, and standards and tools, were investigated using qualitative and quantitative research methods. Some of the challenges identified by the experts indeed match the conclusions of the existing surveys, presented in Chapter 3.

The current work allowed us to rank the existing research according to their importance, as perceived by industry and academia. In particular, the findings in the present work can:

- contribute to the structured and focused further development of various disciplines within the co-simulation community;
- guide the efforts of the scientific community to address problems that are directly relevant to industry; and
- serve as a practical guide by providing references to existing surveys, promising standards and tools for co-simulation.

In the next section, we provide some background on the methodology used. Then, in Section 4.2 we present and discuss the results. Finally, we summarize and conclude in Section 4.3.

4.1 Method and Rationale

In this section, we describe our methodology, the expert selection process, and how the answers were handled.

4.1.1 Delphi Method

As a methodological foundation of this study, the Delphi method [102] was adopted.

The Delphi method is an empirical research method that relies on the systematic compilation of knowledge from a selected group of experts [102, 186, 322, 323]. It fosters the exploration of problems that are characterized by an incomplete state of knowledge [305], a lack historical data, or a lack of agreement within the studied field, which makes it a perfect fit to apply to co-simulation [286]. The aim of applying the Delphi method is to arrive at a reliable shared opinion by means of a repetitive assessment process that includes controlled feedback of opinions [225]. The Delphi method provides structured circumstances that "[...] can generate a closer approximation of the objective truth than would be achieved through conventional, less formal, and pooling of expert opinion" [33].

The Delphi study applied in the present work includes two rounds. The choice of rounds was justified by, for instance, Sommerville [335], who argued that the changes in the participants' views occurred in most cases during the first two rounds of the study and few insights were gained in further rounds. The quality of the Delphi process depends on the factors of creativity, credibility, and objectivity [281]. To address these quality criteria we followed acknowledged guidelines that have been provided by authors such as tose of [225, 281, 286].

The questions in the first round were selected based on the existing studies on co-simulation and the experience of the authors of the current study. Both rounds included qualitative (open-ended) and quantitative questions.

In the first round, the majority of questions asked were qualitative, whereas in the second round, they were quantitative. This ensured that the topic could be introduced in a general way in the first round. To see why, note that if the first round had consisted mainly of quantitative questions, there would have been an increased risk of overlooking important factors or biasing the results.

The qualitative questions asked in the first round only addressed findings that were common among survey papers referred to above. In these cases, expert opinions were used to evaluate the findings of the previous surveys and enable quantitative statements and comparisons to be made. The quantitative questions asked in the second round were mainly formulated based on the results of the first round and the findings reported in the recent literature (e.g., where contradictions were identified).

Regarding the number of experts, Clayton [95] indicated that fifteen to thirty experts with homogeneous expertise backgrounds or five to ten experts with heterogeneous backgrounds should be involved in a Delphi process, while Adler and Ziglio [15] argued that ten to fifteen experts with homogeneous expertise backgrounds could already be considered appropriate.

4.1.2 Expert selection and response rate

The Delphi method does not prescribe any particular way of selecting experts. We used a Knowledge Resource Nomination Worksheet (KRNW) as a framework [286]. The KRNW was proposed in [106] as a general criterion that could be used to sample an expert panel by classifying the experts before selecting them in two iteration steps, to avoid overlooking any important class of experts. This framework consists of the following five steps, detailed below: (1) preparation of the KRNW; (2) population of the KRNW; (3) nomination of additional experts; (4) ranking of experts; and (5) invitation of experts.

In Step (1), we classified the experts according to whether they worked in *academia* or *industry*, as both perspectives were considered essential. Then, in Step (2), the *academia* category was populated based on a keyword-based search in the literature on the state of the art in co-simulation; the *industry* category was populated based on the same keyword-based search and the experience of the authors. Afterwards, in Step (3), both categories were expanded based on the suggestions received after contacting the initial list of experts. In Step (4), the ranking of experts was done using the number of publications in the field of co-simulation, which was obtained from Scopus®[1]. In Step (5) the final group of experts was invited to take part in the Delphi study. Fifteen experts were contacted for the first round; after receiving a final reminder by email, twelve completed questionnaires were returned. The response rate for the first round was, thus, 80 %. In the second round, we contacted seventy persons; after receiving a final reminder by email, 53 completed questionnaires were returned. The response rate for the second round was, thus, 76 %. We can safely state that a significant share of representatives from co-simulation experts were involved in the analysis [15, 95].

Experts from industry who took part in the survey worked in the following sectors: energy Systems (5), software development (7), mobility (4), engineering services (1), system engineering (1), avionics, railways (1). Experts from academia who took part in the survey work in the following fields: energy-related applications (8), software development (6), automotive (3), computer Science (2), maritime (1), system Engineering (1), numerical mathematics (1), system modelling and verification (1) and formal methods (1). Some experts did not provide information about their field or sector.

Table 4.1 summarizes the aim and approach of each round and provides the number of participants per category.

4.1.3 Presentation of the results

A content analysis was performed following the method of Mayring to analyze the qualitative answers [249]. Authors of scientific literature have conducted controversial discussions about which statistical measures are suitable for the interpretation of results of a survey, such as Likert-scales. Hallowell and Gambatese [172] argued that results should be reported in terms of the median rather than the mean, because the median response is less likely to be affected by biased responses. The median is the middle observation in a sorted list of data, separating the upper half from the lower half of a dataset. Sachs [316] argued that the interpolated median is more precise than the normal median, because it is better to consider the frequencies of answers within one category in comparison to all answers. The

[1] www.scopus.com

Table 4.1: Summary of method. Legend: A (Academia), I (Industry), ND (non-Disclosed).

			Participants			
Round	Aim	Approach	A	I	ND	Total
1	Identification of research needs, SWOT factors, limitations and possible extension.	Qualitative	7	2	3	12
2	Evaluation of the result from the first round and development of in-depth discussions on the key aspects. Test on convergence the identified factors, themes and scenarios	Semi-quantitative.	24	19	10	53

interpolated median is used to adjust the median upward or downward within the lower and upper bounds of the Median (M), in the direction in which the data are more heavily weighted. The interpolated median (IM) is calculated as follows:

$$IM = \begin{cases} M & \text{if } n_2 = 0, \\ M - 0.5 + \frac{0.5 \cdot N - n_1}{n_2} & \text{if } n_2 \neq 0 \end{cases} \tag{4.1}$$

where N is the total number of responses to the question, n_1 is the number of scores strictly less than M and n_2 is the number of scores equal to M. For example, if all choices coincide with the median M (that is, $n_1 = 0, n_2 = N$), then the interpolated median coincides with the median. If, on the other hand, no choice is smaller than M (that is, $n_1 = 0$) and there is only one choice that coincides with M, that is, $n_2 = 1$, then IM will be larger than the median. Finally, if most choices are strictly below M (that is, $n_1 = \frac{N}{2} - 1$) and all the other choices coincide with M ($n_2 = 1$), then $IM < M$.

In order to provide a transparent presentation of the results, all results are displayed in detail in a bar chart in Section 4.2, along with their mean, median, and interpolated median, values.

4.1.4 Threats to validity and limitations of the study

Detailed discussion about the threats to validity in Delphi studies can be found in [173]. The selection of experts from academia was done based strictly on the number of publications listed in Scopus®. There is an ongoing discussion about how to compare the scientific impact among researchers. While some indices are well-suited for comparing researchers within the same field, this is not the case for comparing different fields. Since co-simulation is an interdisciplinary field of research, the selection of experts in this work can be seen as a threat to validity. The ranking of experts from industry was done based on the number of publications listed in Scopus®. In addition, we selected experts from industry who we knew have been working with co-simulation for a long time and who have theoretical

and practical knowledge in the field of co-simulation. It can be regarded as a limitation regarding the representativity of the results; however, the responses of the experts indicated that they indeed were well experienced. This selection process ensured that also experts from industry whose focus is not on scientific publishing participated in the study.

4.2 Results and Discussion

In this section, we present the key findings from the Delphi study and the SWOT-AHP analysis.

In the results below, most questions are multiple choice, and the options available were collected during the first round of the Delphi study. To accommodate for additional answers, an extra open field was provided. These open answers, where applicable, are displayed under the *Other* category.

4.2.1 Simulator and Co-simulation Characterization

In order to analyze the purpose for which experts used co-simulation, experts were asked to select the properties that apply to the simulators with which they have worked in co-simulation. As can be seen in Figure 4.1, the majority of the simulators being used in co-simulation represented continuous time simulation units, as defined in Section 3.7. Still, between 18% and 25% of the experts used simulators as "specialized in networks", as "specialized in software controllers", as "a dedicated piece of hardware" or as "receiving input from a human machine interface".

The properties that were not pre-defined in the questionnaire represented a minority of answers: one expert used co-simulation to prove a theorem and another, to solve partial differential equations using finite volume methods. These results indicate that the first round of the Delphi study was successful in that the uses of the simulators could be characterized, and determine that co-simulation was used for many different applications.

4.2.2 Dissemination channels

To identify the main dissemination channels, experts were asked to name the three most important scientific sources used to disseminate their work. The results are shown in Figure 4.2. The Modelica Conference was cited as by far the most important channel for experts used to disseminate their work. The FMI has been one of two key topics in this conference, suggesting that this result is co-related with the fact that the FMI is considered to be the most promising standard for co-simulation (see Section 4.2.3). The dissemination channels suggested by the experts are highly heterogeneous, which underlines the assumption that co-simulation is indeed a multi-disciplinary research field.

4.2.3 Ranking of Standards and Tools

To identify the standards for continuous time, discrete event and hybrid co-simulation, we asked experts (i) to give their opinion on widely accepted standards and describe (ii) what standard they used for co-simulation. The results are summarized in Figure 4.3.

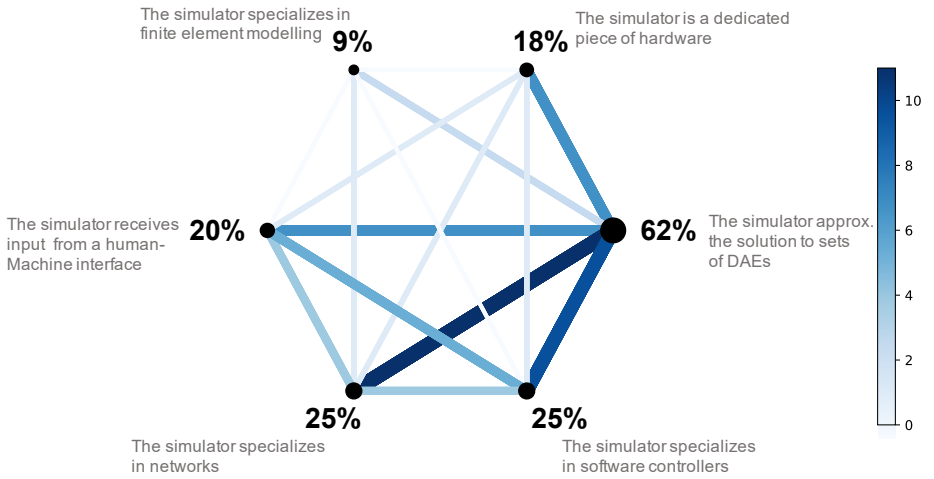


Figure 4.1: Answers to the question: "which properties apply to the simulators . . . ?". Each node represents a property. The size of each node is proportional to the number of positive responses to the corresponding property. Moreover, the thickness of the edge-connections nodes x to nodes y indicates that the same expert gave positive reply to both property x and y. Note that the latter does not imply (and neither neglect) the different properties to apply in one and the same co-simulation.

As can be seen in the figure, the FMI standard is by far the most commonly used standard for any kind of co-simulation.

While the responses for "widely accepted standards" and "standards which experts use" were similar for continuous time and hybrid co-simulation, a different picture emerged for discrete event co-simulation. FMI was described as widely accepted for discrete event co-simulation by 39 % of the experts, however, 68 % of the experts used FMI for discrete event co-simulation. A dedicated empirical study, similar to the one presented here, was performed to identify challenges/barriers to the adoption of the FMI standard [323]. The main results of that study are summarized in Table 4.2.

In addition to promising standards, experts were asked which tools they used for co-simulation. The most common tools used for continuous time co-simulation were Modelica tools and Matlab/Simulink. The use of Modelica tools was cited by 40 % of the experts and about 24 % of the experts mentioned that they used Matlab/Simulink. For discrete event and hybrid co-simulation, no tool was significantly more frequently mentioned than others. The detailed results can be found in Figures 4.4 to 4.6. While only seven different tools were listed for CT co-simulation, thirteen were listed for DE twelve 12 for hybrid co-simulation.

4.2.4 Current challenges

In the first round of the Delphi study, experts commented on current challenges. Based on these responses and the state-of-the-art surveys, we formulated several statements regarding personal experiences. In the second round of the Delphi Study, we posed these statements

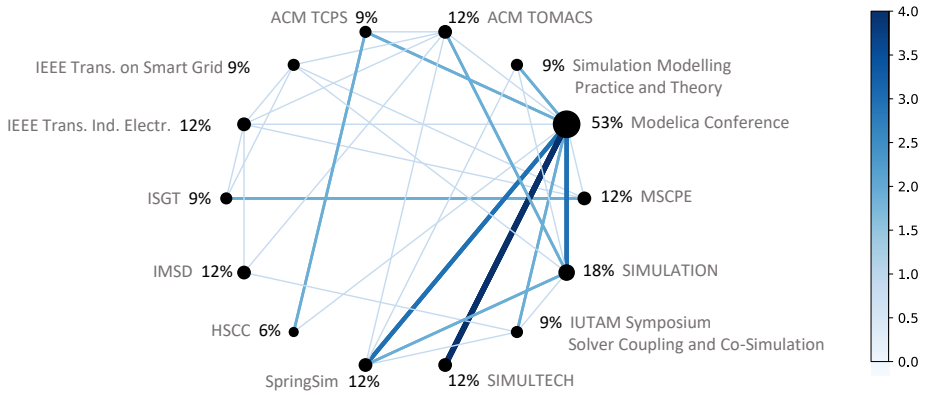


Figure 4.2: Experts were asked to mark the three most important scientific sources they used to disseminate their work. The numbers next to the nodes correspond to the ... % of positive responses; the size of the nodes is also proportional to number of positive responses. The statements upon which the respective experts agreed were connected. MSCPE = Workshop on Modeling and Simulation of Cyber-Physical Energy Systems; ISGT = IEEE Conference on Innovative Smart Grid Technologies; IMSD = International Conference on Multibody System Dynamics; HSCC = Conference on Hybrid Systems: Computation and Control. Conferences mentioned only once by experts are not shown; these included the Conference of the IEEE Industrial Electronics Society, IEEE transactions on power delivery, IEEE Power & Energy Society General Meeting, International Association of Applied Mathematics and Mechanics, Problems in Science and Engineering, European Community on Computational Methods in Applied Sciences and Workshop on Co-simulation of Cyber Physical Systems.

as questions (e.g., "Have you experienced…"). The experts then used a 6-point Likert scale that ranged from from $1 =$ "very frequently" to $6 =$ "never".

Figure 4.8 shows the response count, and Table 4.3 summarizes the responses sorted according to how often experts experience each challenge. A detailed discussion of the individual challenges goes beyond the scope of this survey. However, appropriate references are provided next to each challenge.

The results indicate that practical aspects (as opposed to scientific problems) dominate the problems encountered when conducting co-simulation. The difficulties encountered when judging the validity of a co-simulation present pertinent challenges, already important in the simulation field [109, 336] and aggravated by the black-box nature of co-simulation.

Most challenges (all except simplistic extrapolation functions and difficulties in choosing the correct master algorithm) were assessed by the experts with an interpolated median value greater or equal to four, implying at least occasional occurrence. The experts, thus, confirm the challenges identified in the first round and from the state of the art.

Many experts identified having difficulty choosing the right macro step size, defining

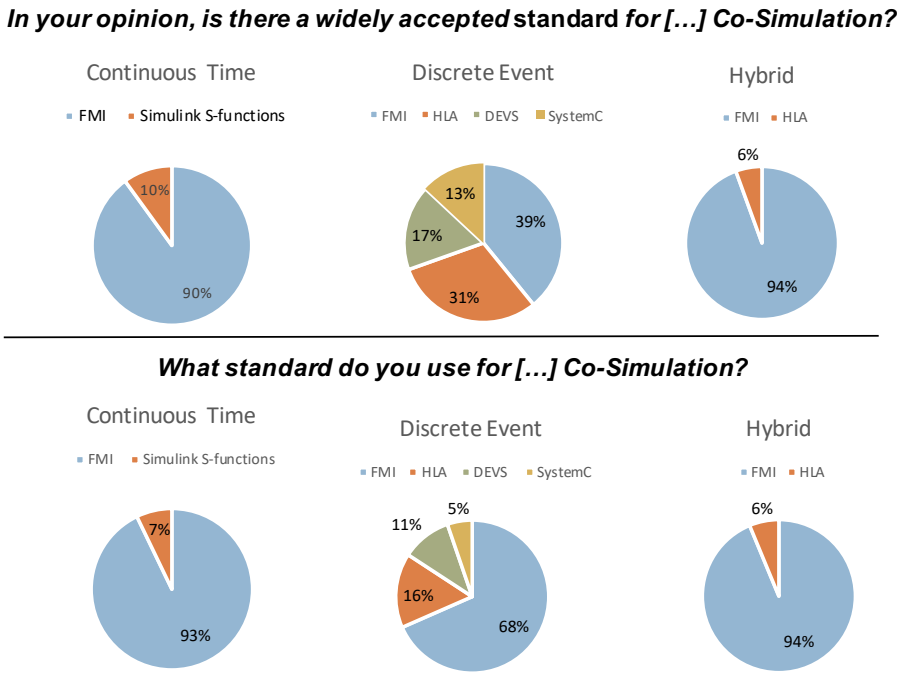


Figure 4.3: Widely accepted and used standards for co-simulation. Depending on the sub figure, the brackets [...] correspond to “Continuous Time”, “Discrete Event” or “Hybrid”

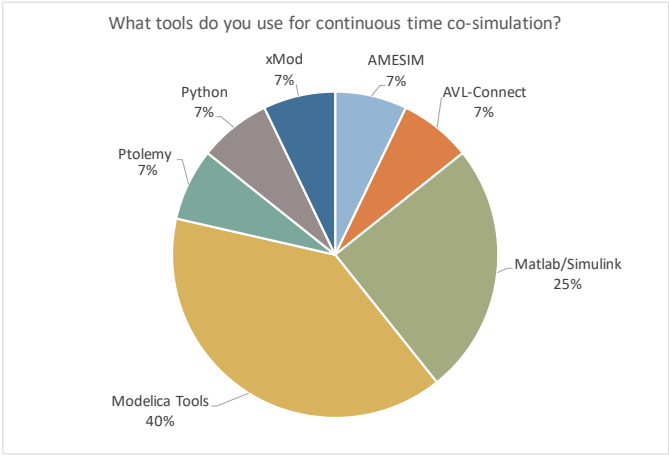


Figure 4.4: Tools that experts use for continuous time co-simulation.

tolerances and with numerical stability. From these responses, we conclude that there is a need for frameworks that provide suitable suggestions to ease the choices for the user. In particular, we assume that many user have significantly less know-how in the areas in practice than the experts interviewed in this work.

Table 4.2: Expert assessment of current barriers for FMI. Based on a Seven-point Likert scale. Modified from [323].

Score: Entirely agree (7) Mostly agree (6) Somewhat agree (5) Neither agree nor disagree (4) Somewhat disagree (3) Mostly disagree (2) Entirely disagree (1)	Mean	Median	Interpolated Median
FMI has limited support for hybrid co-simulation and it is not easily applicable	5.82	5.00	**5.00**
Lack of transparency in features supported by FMI tools	5.12	5.00	**5.05**
There is insufficient documentation and a lack of examples, tutorials, etc.	5.14	5.00	**5.17**
The standard does not support certain requirements that would be widely needed by industry and academia	5.42	5.00	**5.25**
FMI has limited support for discrete co-simulation and it is not easily applicable	5.67	5.00	**5.25**

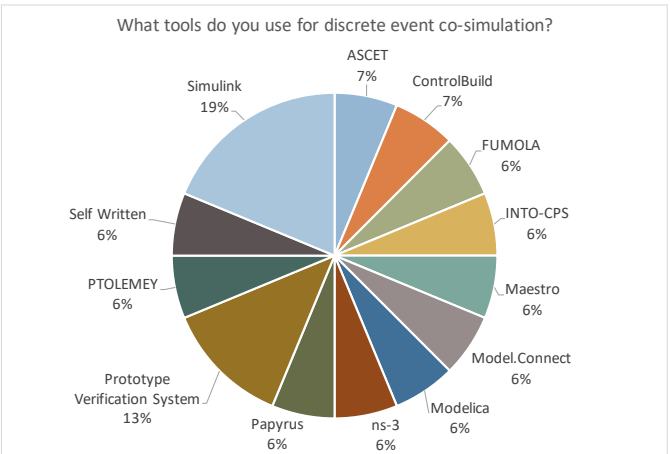


Figure 4.5: Tools that experts use for discrete event co-simulation.

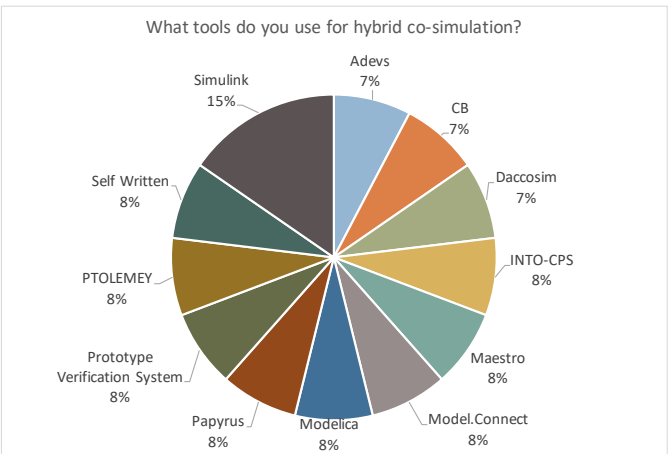


Figure 4.6: Tools that experts use for hybrid co-simulation.

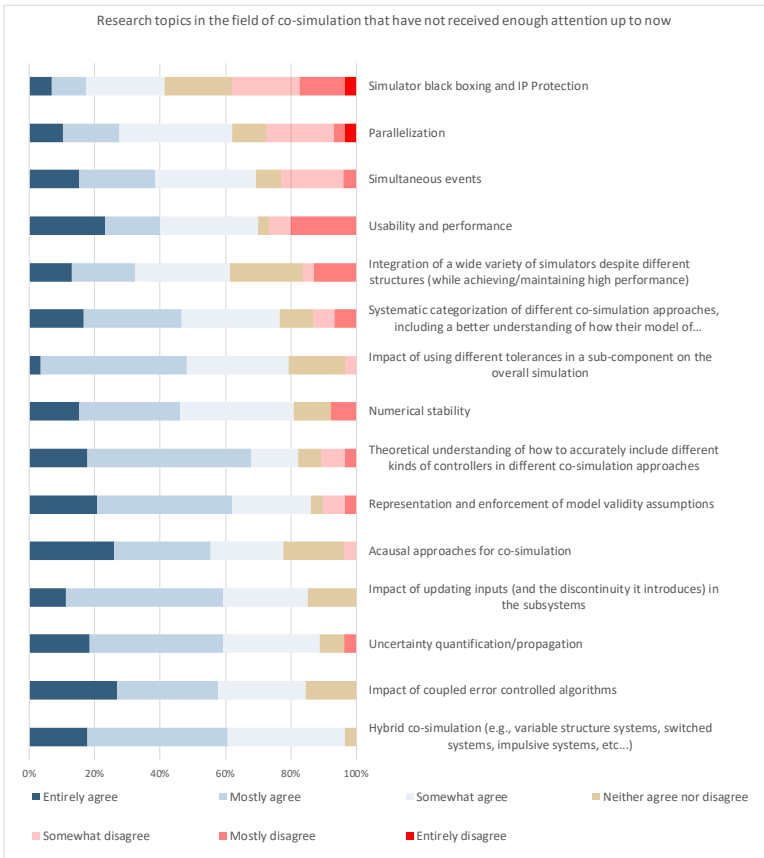


Figure 4.7: Research needs.

4.2.5 Research needs

Experts were asked about research topics in the field of co-simulation that have not received enough attention up until now. Figure 4.7 summarizes the response count on a 7-point scale from “Entirely Disagree” to “Entirely agree”. Furthermore, Table 4.4 shows the same data, sorted in ascending order of topics that *have not* received enough attention until now.

Most research needs (all except simulator black boxing and IP protection) are assessed by the experts with a interpolated median value greater 4.5, corresponding to at least ”Somewhat agree”. Seven research needs were rated with an interpolated median score of greater or equal to 5.5 which corresponds to at least ”Mostly agree”. The experts thus confirm the research needs identified in the first round and from the existing surveys.

In the context of hybrid co-simulation, an expert mentioned that there is only limited awareness about the problems that can arise in hybrid co-simulation; in many cases, it is difficult for user to understand whether problems arise due to shortcomings in standards, tool implementation, or usage. Another expert stressed that the fundamental question of what hybrid co-simulation is and what it should be able to do is a controversial one. Is the

Table 4.3: Experts' assessments: Current challenges. Score: Very Frequently (6) Frequently (5) Occasionally (4) Rarely (3) Very Rarely (2) Never (1).

	Mean	Median	Interp. Median
Difficulties in practical aspects, like IT-prerequisites in cross-company collaboration.	4.7	5.0	4.7
Difficulties due to insufficient communication between theorists and practitioners.	4.4	5.0	4.6
Difficulties in judging the validity of a co-simulation.	4.6	4.0	4.4
Difficulties in how to define the macro step size for a specific co-simulation [42, 73, 157].	4.3	4.0	4.3
Numerical stability issues of co-simulation [23, 71, 158].	4.4	4.0	4.3
Issues with algebraic loops [157, 217].	4.2	4.0	4.2
Difficulties in how to define tolerances.	4.3	4.0	4.0
Issues because of too simplistic extrapolation functions.	3.5	4.0	3.6
Difficulties in choosing the right co-simulation master algorithm.	3.6	3.0	3.4

Table 4.4: Experts assessments: Research needs. Score: Entirely agree (7) Mostly agree (6) Somewhat agree (5) Neither agree nor disagree (4) Somewhat disagree (3) Mostly disagree (2) Entirely disagree (1).

	Mean	Median	Interp. Median
Theoretical understanding of how to accurately include different kinds of controllers in different co-simulation approaches	5.5	6.0	5.9
Representation and enforcement of model validity assumptions [109, 336]	5.6	6.0	5.8
Hybrid co-simulation (e.g., variable structure systems, switched systems, impulsive systems, etc...) [99, 158]	5.8	6.0	5.8
Impact of coupled error controlled algorithms [73, 170]	5.7	6.0	5.8
Uncertainty quantification/propagation [58, 227]	5.6	6.0	5.8
Impact of updating inputs (and the discontinuity it introduces) in the subsystems [71, 153].	5.6	6.0	5.7
Acausal approaches for co-simulation [317]	5.6	6.0	5.7
Impact of using different tolerances in a sub-component on the overall simulation [24]	5.3	6.0	5.5
Numerical stability [72, 154, 155]	5.3	5.0	5.4
Systematic categorization of different co-simulation approaches, including a better understanding of how their model of computations and requirements overlap and differ [348]	5.2	5.0	5.4
Usability and performance	4.9	5.0	5.2
Simultaneous events [65]	5.0	5.0	5.1
Integration of a wide variety of simulators despite different structures (while achieving/maintaining high performance) [156]	4.8	5.0	4.9
Parallelization [319, 349]	4.6	5.0	4.9
Simulator black boxing and IP Protection [47]	4.1	4.0	4.1

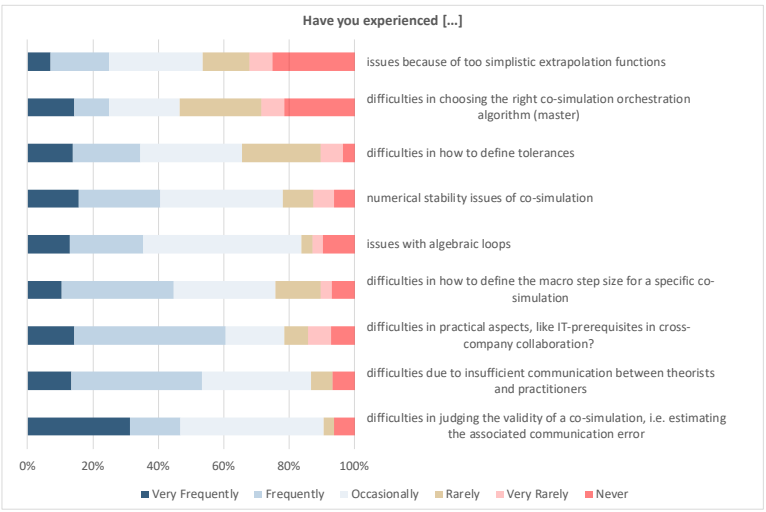


Figure 4.8: Current challenges.

intention to allow the same flexibility with hybrid co-simulation as there is in monolithic simulation (with everything that this entails) or is the intention to couple large subsystems? This expert concluded that the two different views have different requirements with respect to hybrid co-simulation, and this is a largely unexplored topic that needs more research regarding numerical properties.

4.3 Concluding Remarks

This chapter presents an expert assessment on co-simulation, addressing the social and empirical aspects and placing a focus on promising standards and tools, current challenges and research needs. As a methodological foundation of this study, the Delphi method was adopted.

The authors consider the following findings from the empirical data as the most important:

- Experts consider the FMI standard as the most promising standard for continuous time, discrete event and hybrid co-simulation;
- Experts frequently have difficulties dealing with practical aspects, like IT-prerequisites in cross-company collaboration, and encounter problems due to insufficient communication between theorists and practitioners.
- The most important research needs identified by experts are: (i) theoretical understanding of how to accurately include different kinds of controllers in different co-simulation approaches, (ii) validity aspects, (iii) hybrid co-simulation (iv) accuracy aspects and (v) acausal approaches;
- The highest ranked difficulty relates to practical aspects while the highest ranked research need related to theoretical understanding. This is not a contradiction; this insight may help for making co-simulation easier to use in practice;
- The results of the SWOT-AHP analysis indicate that factors for strengths and op-

portunities predominate. The experts assign the highest important to the need for user-friendly tools including pre-defined master algorithms, integrated error estimation, etc.;

Statistical tests were conducted to determine differences in the perceptions of experts from industry and academia regarding the current challenges and open research topics; no significant difference were observed. We refrained from testing more complex hypotheses in this study, due to the number of answers and the non-probability sampling approach taken.

Limitations. There is a potential bias towards the FMI standard, as the majority of experts answered that this is the standard they use for co-simulation. Moreover, the definition of co-simulation used in this survey was very narrow (i.e., a master algorithm is one of Jacobi, Gauss-seidel, or Strong-coupling), and it is not the definition that we use throughout this thesis. This mismatch may cause confusion when interpreting the results.

It is our hope that the results of this study will increase transparency and facilitate the structured development of co-simulation standards and tools. They influenced our research by showing us that stability analysis, configuration of co-simulations, and the state event location configuration, are pertinent research problems (recall Figure 4.8). These are explored in the following chapters.

Chapter 5

Stability Preservation in Adaptive Co-simulation

Disclaimer The content in this chapter is adapted from:

- GOMES, CLÁUDIO, Benoît Legat, Raphaël M. Jungers, and Hans Vangheluwe. "Stable Adaptive Co-Simulation: A Switched Systems Approach." In IUTAM Symposium on Co-Simulation and Solver Coupling. Darmstadt, Germany, 2017.
- GOMES, CLÁUDIO, Benoît Legat, Raphaël Jungers, and Hans Vangheluwe. "Minimally Constrained Stable Switched Systems and Application to Co-Simulation." In IEEE Conference on Decision and Control. Miami Beach, FL, USA, 2018.

This chapter marks the second part of this thesis. We shift from trying to understand the fundamental challenges in co-simulation, into exploring possible solutions.

5.1 Introduction

Co-simulation promotes the idea that each simulator decides how to best compute the behavior of the subsystem allocated to it, leaving to the master algorithm the decision of when (with respect to the simulated time) should the simulators exchange data, and in what order [158]. However, as Section 3.7, and prior work referred to in that section, show, the decision on how to best compute the behavior of each sub-model depends on the specific arrangement of all simulation units, and on the decisions of the master. In sum: no decision concerning how to compute the co-simulation should be taken independently of the co-simulation scenario, which means that simulators should avoid "hard-coded" decisions.

It is currently a matter of research to find out which decisions are scenario dependent, and in this chapter, we assume that each simulator provides a mechanism to control some of these. Two factors are known to affect these decisions: (1) the co-simulation scenario (i.e., sub-models and how they are connected); and (2) the requirements for the co-simulation (i.e., performance, accuracy, etc).

Regarding the exact moment when these decisions need to be made, in the general case of systems that undergo structural changes (and therefore change the co-simulation scenario), the only possible time to make such decisions is when these changes occur, as demonstrated in [260]. The requirements for the co-simulation can change during the computation itself as well. The purpose of this is to inspect certain transient behavior of interest (e.g., see [37, 182, 204, 312]). We will therefore focus on adaptive co-simulation, where the master and simulators change the way they compute the co-simulation during the co-simulation itself, as a factors (1) and (2) change.

The ability to adapt allows one to find the best tradeoff between accuracy and computation power, to meet the available time. The applications are not restricted to co-simulation. For example, in Model Predictive Control [137], the controller needs to be able to simulate the system in real-time.

In the scope of adaptive co-simulation, it is hard to predict which decisions are to be taken without actually computing the co-simulation. It is then natural to wonder whether it is possible to ensure trustworthy co-simulation results, in the face of such uncertainty.

5.1.1 Contribution

In this chapter, we focus on the numerical stability of adaptive co-simulation algorithms. We show how to prove that a co-simulation algorithm is numerically stable, provided that the set of all possible decisions (i.e., reactions to changes in factors (1) and (2)) is known.

We will call *policy sequence* to the sequence of policies taken by all simulators over time. If there exists one or more policy sequences that can make the (co-) simulation method unstable, then the simulators have to be forbidden from following these. We first show a naive approach to this problem, and then develop a better one, that minimizes the cost of opportunity for forbidding policy sequences.

In particular, we propose to use the joint spectral radius theory [197] to represent the co-simulation global error as switched system (Section 5.10). Hence the preservation of stability becomes a problem of deciding the stability of a switched system. Furthermore, as the choice of future policies is often influenced by the past policies, we consider *constrained switched systems*, a recently developed framework allowing us to model the memory of the system (see Section 5.3).

If there exist one or more policy sequences that can make the (co-) simulation method unstable, then the simulators have to be forbidden from following these. In the context of constrained switched systems, there are many ways of forbidding a policy sequence, and each way also forbids sequences that do not make the (co-) simulation unstable. We discuss this optimization problem and propose an algorithm to solve it, based on the concept of entropy.

Finally, we provide an open source library that comprises the algorithms developed in this chapter, and discuss it's implementation. This library uses the results in [233, 234].

5.1.2 Structure

In the next section, we experiment with some examples. Then, in Section 5.3, we introduce the concepts that will help us make sense of the problem we are trying to solve. In Section 5.4 we discuss how to prove the stability of an adaptive co-simulation schemes, and then in Section 5.5 we analyze the problem of how to stability an unstable adaptive scheme, and describe a naive solution. In Section 5.6 we propose an algorithm that approximates the solution, and we prove that the it terminates and that the resulting system is stable. Furthermore, we provide a lifting technique that yields better solutions, and we propose an implementation in Sections 5.7 and 5.8. Finally, we discuss the optimality of our solution in Section 5.9, present experimental results in Section 5.10, related work in Section 5.11, and conclude in Section 5.12.

5.2 Motivational Examples

We motivate our work using two well known examples: one in the simulation domain, and one in the co-simulation domain.

5.2.1 Adaptive Simulation

Consider the problem of approximating the solution $x(t)$ of the system,

$$\dot{x}(t) = \bar{A}x(t), \text{ with } x(0) = x_0, \tag{5.1}$$

using an adaptive simulation algorithm. These methods are useful in situations where, e.g., the error tolerance, or run-time performance, can vary as a function of $\bar{x}(t)$ and t [87, 141, 330]. In practice, multi-step variable order methods [87, Section 4] are the most commonly used, but for illustrative purposes, we show a single step method. The analysis we describe in the following sections can be applied to multi-step variable order methods.

Example 19. The approximation $\tilde{x}(t)$ of the solution to System (5.1), computed by a simulation algorithm that uses different step sizes, and different numerical methods, can be modeled as a switched system, formally defined in Section 5.3, with

$$\begin{aligned}
x_{i+1} &= A_{\sigma(i)}x_i : \ \ \sigma(i) \in \{0, \dots, m-1\}, \ A_{\sigma(i)} \in \mathcal{A} \\
\mathcal{A} &= \left\{ \tilde{A}_{fe,h}, \tilde{A}_{md,h}\tilde{A}_{rg,h} \middle| h \in \{0.001, 0.002\} \right\} \\
\tilde{A}_{fe,h} &\triangleq \mathbf{I} + \bar{A}h \\
\tilde{A}_{md,h} &\triangleq \mathbf{I} + \bar{A}h + (\bar{A}h)^2/2 \\
\tilde{A}_{rg,h} &\triangleq \mathbf{I} + \bar{A}h + (\bar{A}h)^2/12 + (\bar{A}h)^3/6 + (\bar{A}h)^4/24,
\end{aligned}$$

where x_0 is given, $\{0, \dots, m-1\}$ is the set of modes, $\sigma(i)$ is the mode active at step i, the matrices in $\mathcal{A}$, correspond respectively to the Forward Euler method, the Midpoint method and the Runge-Kutta method.

In Example 19, if one assumes that System (5.1) is stable, that is,

$$\lim_{t \to \infty} \|x(t)\| = 0 \text{ for any } x(0),$$

then we need to ensure that the error made by discrete approximation $\tilde{x}(t)$ is dissipated. For this purpose, we introduce the *error switched system*, whose state variable is $e_t = \tilde{x}(t) - x(t)$, and can be written as

$$\begin{aligned} e_{t+h} &= A_{\sigma(t)} e_t + L_{\sigma(t)}, \text{ with} \\ L_{\sigma(t)} &\triangleq \big(A_{\sigma(t)} - \exp(\bar{A}h)\big)x(t), \end{aligned} \tag{5.2}$$

where $L_{\sigma(t)}$ is the local error corresponding to $A_{\sigma(t)} \in \mathcal{A}$, and $\mathcal{A}$ is defined in Example 19. Neglecting $L_{\sigma(t)}$ yields a switched system, the stability of which determines the dissipation of error.

As an example, the error of the adaptive simulation algorithm introduced in Example 19 may not be stable for a policy sequence $111\ldots$, but may be stable for $2121\ldots$ As a result, the adaptive simulation method may opt for a policy sequence that requires fewer model evaluations (compared to a non-adaptative algorithm), whilst preserving the stability. Another way of stating this is to observe that the spectral radius of $\tilde{A}_1$ is larger than 1, that is, $\rho(\tilde{A}_1) > 1$. This *does not* imply that the product of any policy sequence of the form $2121\ldots$ causes the system to be unstable. Figure 5.1 illustrates this fact.

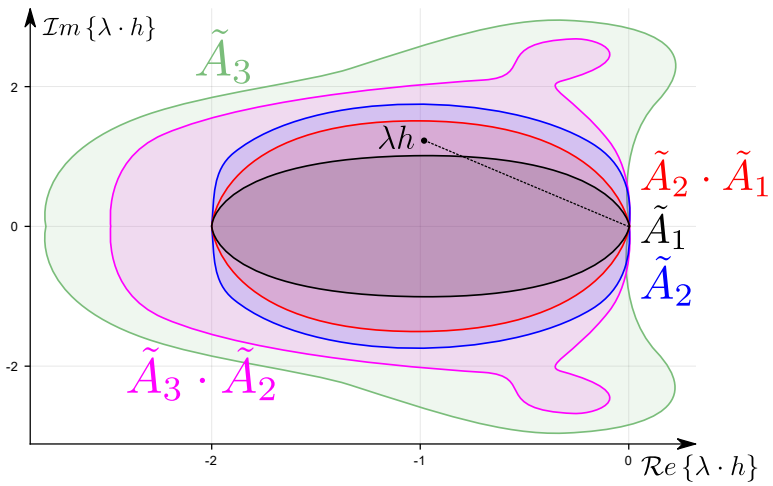


Figure 5.1: Domain of numerical stability for some hybrid methods in $\mathcal{A}^2$, defined in Example 19. See [87, Section 2.4] for an example of how to construct the stability domain. Matrix $\tilde{A}_1$ corresponds to the Forward Euler solver. Matrix $\tilde{A}_2$ corresponds to the Midpoint solver, and $\tilde{A}_3$ to the Runge-Kutta solver. Matrix $\tilde{A}_2 \cdot \tilde{A}_1$ represents a hybrid solver every other simulation step is done by the Forward Euler followed by the Midpoint solver. The matrices are defined in Example 19. Each line corresponds to the product of the step size and Eigen value for which the corresponding solver is marginally stable. The shaded area represents the range of step sizes for which the corresponding solver is stable. As the figure shows, it is possible that, for a given h and Eigenvalue λ of $\bar{A}$, λh is located outside the stability domain of $\tilde{A}_1$ (shaded area in black), but inside the stability region of the hybrid method $\tilde{A}_2\tilde{A}_1$ (shaded in red). In that case, forbidding any policy sequence of the form $11\ldots$, may ensure that the system introduced in Example 19 is stable.

5.2.2 Adaptive Co-simulation

We motivate our work for adaptive co-simulation using a simple and well known system, that is commonly used to study the numerical stability of multiple master algorithms (see, e.g., [23, 71, 72, 74, 201, 216, 325]), and is presented in detail in Section 3.7.

A coupled mass-spring-damper system is shown in Figure 5.2. We consider two simulators—S_1, S_2—and the allocation depicted in the figure: simulator S_1 computes the behavior of the left-hand-side (LHS) mass, accepting the input coupling force F_c, and producing the position and velocity of the mass as outputs; and S_2 accepts the position and velocity computed by S_1, and produces the coupling force F_c. They are coupled as shown in Figure 5.3.

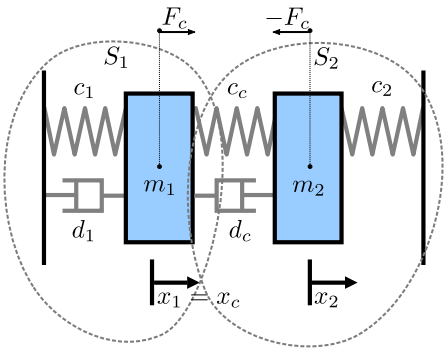


Figure 5.2: Example double mass-spring-damper system.

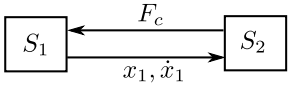


Figure 5.3: Example arrangement of simulators.

The dynamics of the LHS mass are given by:

$$\begin{aligned} &\dot{x}_1(t) = v_1(t); && m_1 \cdot \dot{v}_1(t) = -c_1 \cdot x_1(t) - d_1 \cdot v_1(t) + F_c(t); \\ &x_1(0) = p_1; && v_1(0) = s_1. \end{aligned} \tag{5.3}$$

where $\dot{x}$ denotes the time derivative of x; c_1 is the spring stiffness constant and d_1 the damping coefficient; m_1 is the mass; p_1 and s_1 the initial position and velocity, respectively; and $F_c(t)$ the input force acting on the mass over time.

The right-hand-side mass is governed by:

$$\begin{aligned} &\dot{x}_2(t) = v_2(t); && m_2 \cdot \dot{v}_2(t) = -c_2 \cdot x_2(t) - F_c(t); \\ &x_2(0) = p_2; && v_2(0) = s_2; \\ & && F_c(t) = c_c \cdot (x_2(t) - x_1(t)) + d_c \cdot (v_2(t) - v_1(t)); \end{aligned} \tag{5.4}$$

where c_c and d_c denote the stiffness and damping coefficients of the central spring and damper, respectively; c_2 denotes the stiffness constant for the right spring; p_2 and s_2 the initial position and velocity.

We assume that the co-simulation of this example is computed with a Jacobi master algorithm, as summarized in Algorithm 4. The function DoStep(H,S) instructs simulator S to simulate the behavior of its allocated subsystem in the time interval $t \to t + H$, using an input extrapolation scheme.

ALGORITHM 4: Jacobi master algorithm for the simulators shown in Figure 5.3. This algorithm is defined in Section 3.7.2.

Data: The stop time t_f and a communication step size $H > 0$.

$t := 0$;
while $t \leq t_f$ **do**
  $\begin{bmatrix} x_1 & v_1 \end{bmatrix}^T :=$ GETOUTPUT(S_1);
  SETINPUT(S_2, $\begin{bmatrix} x_1 & v_1 \end{bmatrix}^T$);
  $F_c :=$ GETOUTPUT(S_2);
  SETINPUT(S_1,F_c);
  DOSTEP(H,S_1);
  DOSTEP(H,S_2);
  $t := t + H$;
end

Figure 5.4 shows multiple co-simulations of the system in Figure 5.2, using different configurations for the simulators.

Comparing the plotted trajectories, we see that there is something wrong with trajectory `x1_fer1`. Due to the positive damping constants, the original system must always come to a rest, irrespective of the initial values. However, the co-simulation that produces `x1_fer1` does not seem to satisfy this property.

To compare the performance of each co-simulation, we compute the number of model evaluations. For the co-simulations producing the trajectories `x1_fer10` and `x1_fer1`, this is given as:

$$\frac{t_f}{H} \times \left(\mathsf{Steps}_{S_1} + \mathsf{Steps}_{S_2}\right),$$

where t_f is the maximum simulation time, and Steps_S denotes the number of internal integration steps performed by simulator S, per invocation of DOSTEP(H,S). The algorithm that computes trajectory `x1_ferm` is designed to spend 70% of the time using the parameters used to compute `x1_fer10` and the remaining time using the parameters used to compute `x1_fer1`. It gives the following evaluations:

$$0.7 \times \mathsf{Evals}_{\mathtt{cs}_1} + 0.3 \times \mathsf{Evals}_{\mathtt{cs}_2}.$$

As can be seen in Table 5.1, the adaptive co-simulation mimics the qualitative behavior of the system (i.e., eventually coming to a rest), with fewer model evaluations than `x1_fer10`.

Table 5.1: Total number of model evaluations per co-simulation in Figure 5.4.

Trajectory	Evaluations
`x1_fer10`	20000
`x1_fer1`	2000
`x1_ferm`	14600

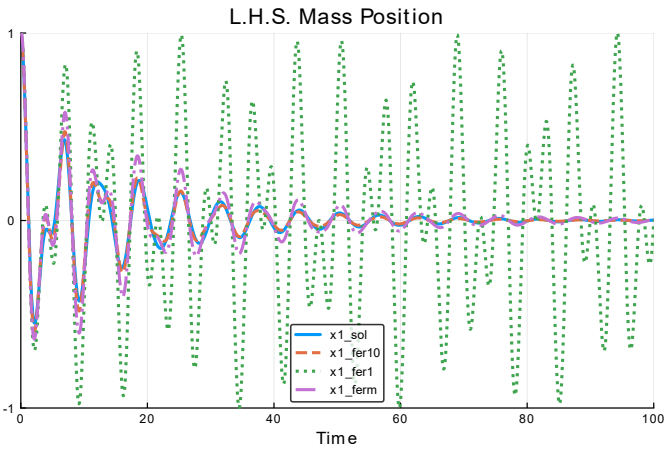


Figure 5.4: LHS mass position co-simulations of the system in Figure 5.2. Parameters: $m_1 = c_1 = m_2 = c_2 = c_c = 1.0$; and $d_1 = d_c = 0.1$. The co-simulation step used is $H = 0.1$. Trajectory `x1_sol` denotes the correct trajectory of $x_1(t)$, for reference, obtained by coupling Equations (5.3) and (5.4) and finding the analytical solution; `x1_fer10` denotes the trajectory obtained with a co-simulation where both simulators employ the forward Euler method, using a constant extrapolation of the inputs, and performing 10 internal integration steps per co-simulation step; `x1_fer1` is similar to `x1_fer10`, except each simulator performs only one integration step per co-simulation step; `x1_ferm` is obtained with a co-simulation that adaptively combines the configuration used in `x1_fer10` and `x1_fer1`, i.e., it varies the number of internal integration steps per simulator.

This minimal example highlights one of the advantages of adaptive co-simulations: the ability to obtain better tradeoffs between mimicking the qualitative behavior of the original system, and performance.

Consider now the adaptive co-simulation `x1_fer1M` shown in Figure 5.5, which is similar to the policy used to compute `x1_ferm`, except that more time is spent in the mode where the simulators only take one integration step. Despite being adaptive, it does not seem to come to a rest, which brings to **our research problem**: how can we discern a stable adaptive co-simulation, from an unstable one?

The next section provides the necessary background to explore this problem in depth.

5.3 Background

5.3.1 (Numerical) Stability

In this section, we recall the concepts of numerical stability, introduced in Section 3.7. For convenience, we repeat some of the equations presented before.

Consider the following initial value problem:

$$\dot{x} = Ax; \qquad x(0) \text{ is given;} \tag{5.5}$$

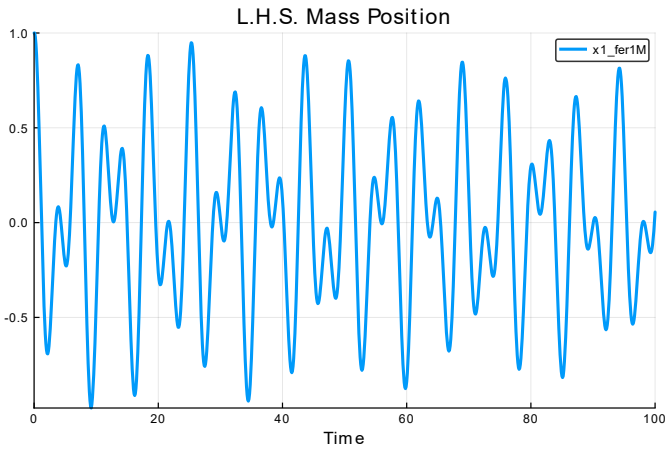


Figure 5.5: Example wrong adaptive co-simulation. Trajectory `x1_fer1M` is similar to the policy used to compute `x1_ferm`, except that more time is spent in the mode where the simulators only take one integration step.

where $x(t)$ is a real-valued vector, and A is a square real matrix.

The solution $x(t)$ to the system in Equation (5.5) is stable if $x(t)$ tends to the origin, regardless of the initial value. In other words, $\lim_{t\to\infty} \|x(t)\| = 0$, for any given $x(0)$.

Suppose that the solution to Equation (5.5) is approximated by the following discrete time system:

$$\tilde{x}_{i+1} = \tilde{A}\tilde{x}_i; \qquad \tilde{x}_0 = x(0); \tag{5.6}$$

where $\tilde{x}_i$ is a real-valued vector, and $\tilde{A}$ is a square real matrix.

We say that the system in Equation (5.6) is stable if, for all $\tilde{x}_0$,

$$\lim_{i\to\infty} \left\|\tilde{x}_i\right\| = 0 \Leftrightarrow \lim_{i\to\infty} \left\|\tilde{A}^i\right\| = 0, \tag{5.7}$$

for any vector norm $\left\|\tilde{x}\right\|$, and any matrix norm $\left\|\tilde{A}\right\|$ satisfying the *submultiplicativity* property.

Definition 15 (Numerical Stability)**.** Assuming that the system in Equation (5.5) is stable, it is important that the approximating system in Equation (5.6) preserves this property, in which case, we denote it as being numerically stable.

If the system in Equation (5.6) is numerically stable, the errors introduced are not amplified.

The condition in Equation (5.7) can be studied by means of the spectral radius $\rho(\tilde{A})$ [341, Theorem 1.3.2]:

$$\rho(\tilde{A}) < 1 \Leftrightarrow \lim_{i\to\infty} \left\|\tilde{A}^i\right\| = 0,$$

where $\rho(\tilde{A})$ is given by Gelfand's formula or the maximum absolute eigenvalue:

$$\rho(\tilde{A}) = \lim_{i\to\infty} \left\|\tilde{A}^i\right\|^{1/i} = \max_j |\lambda_j|, \tag{5.8}$$

and λ_j is the j-th eigenvalue of $\tilde{A}$.

The numerical stability of a co-simulation is analyzed by assuming that the underlying coupled system can be written as in Equation (5.5) and is stable, and computing the discrete time induced model in the form of Equation (5.6) that represents the co-simulation. Here, we illustrate how this is done for the example in Figure 5.2. This procedure can be generalized to any number of simulators, as long as the underlying coupled system can be written as in Equation (5.5).

Consider now the example of Figure 5.2, and suppose that the master and simulators are at time t_i. In the interval $t \in [t_i, t_{i+1}]$, each simulator S_j, with $j = 1, 2$, is trying to approximate the solution to a linear ODE,

$$\begin{aligned} \dot{x}_j &= A_j \cdot x_j + B_j \cdot u_j \\ y_j &= C_j \cdot x_j + D_j \cdot u_j \end{aligned} \tag{5.9}$$

where A_j, B_j, C_j, D_j are matrices with appropriate dimensions, and the initial state $x_j(t_i)$ is the state computed in the most recent co-simulation step. We assume that either D_1 or D_2 is the null matrix, so that the coupled system can be written as Equation (5.5). In this example, $D_1 = \mathbf{0}$.

Without loss of generality (for more sophisticated input extrapolation techniques, see [71, Equation (9)]), we assume that each simulator uses a constant extrapolation to approximate the input in the interval $[t_i, t_{i+1})$. That is, $\tilde{u}_j(t) = u_j(t_i)$, for $t \in [t_i, t_{i+1})$. Then, Equation (5.9) can be re-written to represent the unforced system being integrated by each simulator:

$$\begin{bmatrix} \dot{x}_j \\ \dot{\tilde{u}}_j \end{bmatrix} = \begin{bmatrix} A_j & B_j \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \cdot \begin{bmatrix} x_j \\ \tilde{u}_j \end{bmatrix} \tag{5.10}$$

We can represent the multiple internal integration steps of Equation (5.10), performed by the simulator S_j in the interval $t \in [t_i, t_{i+1}]$, as

$$\begin{bmatrix} x_j(t_{i+1}) \\ \tilde{u}_j(t_{i+1}) \end{bmatrix} = \tilde{A}_j^{k_j} \cdot \begin{bmatrix} x_j(t_i) \\ \tilde{u}_j \end{bmatrix} \tag{5.11}$$

where $\tilde{A}_j$ represents a single integration step of the numerical method (e.g., $\tilde{A}_j = \mathbf{I} + h_j \begin{bmatrix} A_j & B_j \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$ for the forward Euler method), $k_j = (t_{i+1} - t_i)/h_j$ is the number of internal steps, and $0 < h_j \leq H$ is the internal fixed step size that divides H. Note that Equation (5.25) represents a discrete time system modeling the behavior of the simulator at a single co-simulation step, with no inputs. Now we just have to represent how the simulators exchange data at the end/beginning of a co-simulation step.

At the beginning of the co-simulation step i, we wish to enforce $u_1(t_i) = y_2(t_i)$ and $u_2(t_i) = y_1(t_i)$. This, together with Equation (5.9), gives,

$$\begin{aligned} u_1(t_i) &= C_2 \cdot x_2(t_i) + D_2 C_1 \cdot x_1(t_i). \\ u_2(t_i) &= C_1 \cdot x_1(t_i) \end{aligned} \tag{5.12}$$

Finally, Equations (5.10) to (5.12) are combined to write the co-simulation step in the form of Equation (5.6) as

$$\begin{bmatrix} x_1(t_{i+1}) \\ x_2(t_{i+1}) \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \tilde{A}_1^{k_1} & \mathbf{0} \\ \mathbf{0} & \tilde{A}_2^{k_2} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & C_2 \\ \mathbf{0} & \mathbf{I} \\ C_1 & D_1 \cdot C2 \end{bmatrix}}_{\tilde{A}} \begin{bmatrix} x_1(t_i) \\ x_2(t_i) \end{bmatrix} \tag{5.13}$$

whose stability is easily checked with Equation (5.8).

We remark that Equation (5.13) represents an abstraction of how the co-simulation is computed, for analysis purposes. In practice, the co-simulation itself may include optimizations, parallelism, etc... which are neglected when building Equation (5.13).

5.3.2 Joint Spectral Radius

The definitions we present here are adapted from [197].

Consider the following switched discrete time system:

$$x_{i+1} = A_{\sigma(i)} x_i : \ \sigma(i) \in \{0, \ldots, m-1\}, \ A_{\sigma(i)} \in \Sigma \tag{5.14}$$

where x_0 is given, $\{0, \ldots, m-1\}$ is the set of modes, $\sigma(i)$ is the mode active at step i, and $\mathcal{A} = \{A_1, \ldots, A_m\} \subseteq \mathbb{R}^{n \times n}$ is a set of real matrices.

Switched systems are widely used to model many dynamical systems in modern engineering including viral mutations in a patient's body [176], *trackability* of malicious agents in a sensor network [200], or scheduling of thermostatically controlled loads (TCLs) [279].

We denote the sequence $\sigma(0), \sigma(1), \ldots$ as the *switching signal*, where $A_{\sigma(i)} \in \Sigma$ represents the matrix used to compute x_{i+1} from x_i in Equation (5.14). A switching signal $\sigma(0), \sigma(1), \ldots, \sigma(i)$ induces the matrix product $A_{\sigma(i-1)} \ldots A_{\sigma(1)} \cdot A_{\sigma(0)}$. Let

$$\Sigma^i = \left\{ A_{p_{i-1}} A_{p_{i-2}} \ldots A_{p_0} : A_{p_j} \in \Sigma, 0 \leq p_j < m, j = 0, \ldots, i-1 \right\}$$

be the set of all products induced by switching signals with length i. Note that, for any given switching signal $\sigma(0), \sigma(1), \ldots \sigma(i-1)$, $x_{i+1} = Ax_0$ for some $A \in \Sigma^i$.

The system in Equation (5.14) is stable if, for any x_0, and any switching signal, $\lim_{i \to \infty} \|x_i\| = 0$.

The Joint Spectral Radius $\hat{\rho}(\Sigma)$ (JSR) is essentially a generalization of Gelfand's formula, in Equation (5.8), to arbitrary products of matrices in Σ [314]:

$$\begin{aligned} \hat{\rho}_i(\Sigma) &= \sup \left\{ \|A\|^{1/i} : A \in \Sigma^i \right\} \\ \hat{\rho}(\Sigma) &= \limsup_{i \to \infty} \hat{\rho}_i(\Sigma) \end{aligned} \tag{5.15}$$

Using the JSR, we can characterize the stability of the system in Equation (5.14) by noting that [197, Theorem 1], for any bounded set Σ,

$$\hat{\rho}(\Sigma) < 1 \Leftrightarrow \text{ for all } \sigma, \lim_{i \to \infty} \left\| A_{\sigma(i)} A_{\sigma(i-1)} \cdots A_{\sigma(0)} \right\| = 0. \tag{5.16}$$

To determine whether the system in Equation (5.14) is stable, note that the limit in Equation (5.15) exists, and any finite i satisfies:

$$\hat{\rho}(\Sigma) \leq \hat{\rho}_i(\Sigma) \text{ [197, Lemma 1.2].}$$

Therefore, if there exists i, such that $\hat{\rho}_i(\Sigma) < 1$, then the switched system is stable. Note however, that checking whether $\hat{\rho}(\Sigma) < 1$ is undecidable in general [197, Proposition 2.9], and in the cases where we know it is decidable, it is NP-Hard [49].

Other algorithms exist to estimate $\hat{\rho}(\Sigma)$, and we refer the reader to [164, 167, 198, 244, 293].

5.4 Stability Certification of Adaptive Co-simulations

In this section, we first describe how to use the concepts introduced in the previous section to determine the numerical stability of an adaptive co-simulation. Then, we propose a way to address the case when the adaptive co-simulation is not numerically stable.

5.4.1 Stability

Equation (5.13) represents a single co-simulation step, which, as we later shown in Equations (5.23) to (5.26), represents a specific: system arrangement; coupling approach; simulator input approximation; internal solver method; internal simulator step size h_j; and communication step size H. If any of these items changes from one co-simulation step to the next, the co-simulation is adaptive, and is best described as a discrete time switched system, of the form of Equation (5.14), where Σ includes every possible variation of the matrix $\tilde{A}$ in Equation (5.13), constructed as explained in Section 5.3.1.

To exemplify, in the co-simulation of the system in Figure 5.2, suppose that the decision space is as follows:

Arrangement is the one in Figure 5.3;
Coupling is the one in Algorithm 6 but a Gauss-Seidel, Strong coupling, or others, could have been used [165];
Input Approximation is the constant extrapolation but higher order input approximations can be applied [71];
Solver can be forward Euler, or the midpoint method [373, Section II.1];
Communication Step Size H is 0.1;
Solver Step Size can be $H/10$ or H;

Then Σ contains 16 matrices, representing every possible combination of policies, per simulator, from one co-simulation step to the next.

Applying the result in Equation (5.16) ensures that any possible decision sequence taken by the co-simulation always produces a numerically stable co-simulation. This is a strong result in the sense that we do not need to know anything about how the decisions are made (to ensure stability preservation).

In the example proposed, $\hat{\rho}(\Sigma) \geq 1$, which means that there is a switching signal (always use $A_{\text{cs\_2}}$ to compute the next co-simulation step) that causes the co-simulation to not be stable. In fact, the result is the trajectory `x1_cs_2`, plotted in Figure 5.4. To see why

$\hat{\rho}(\Sigma) \geq 1$ holds, let $A_{\text{cs\_2}}$ denote that co-simulation step matrix that uses $H = 0.1$ and solver step size equal to H. Then, computing the spectral radius $\rho(A)$, one observes that $\rho(A) > 1$.

5.4.2 Stabilization

As the paragraph above shows, if there is a matrix $A \in \Sigma$ such that $\rho(A) \geq 1$, then we have that $\hat{\rho}(\Sigma) \geq 1$. This immediately suggests an procedure to be done before computing the JSR: exclude all unstable matrices. That is, we set

$$\Sigma_0 = \Sigma \setminus \{A\}, \forall A \in \Sigma : \rho(A) \geq 1.$$

After computing Σ_0, it can still be the case that $\hat{\rho}(\Sigma_0) \geq 1$, as the product of stable matrices is not necessarily stable (see, e.g., [197, Figure 1.2]). Furthermore, $\hat{\rho}(\Sigma_0) \geq 1$ does not imply that there exists a finite i and a $A \in \Sigma_0^i$ such that $\rho(A) \geq 1$ (see, e.g., [197, Section 2.4], with the case that $\rho(A) = 1$). This means that no algorithm can always ensure that a stable co-simulation is attained (undecidability). Fortunately, in practice, the algorithm proposed in [233] works well.

The work in [233] approximates $\hat{\rho}(\Sigma_0)$, allowing us to check whether $\hat{\rho}(\Sigma_0) < 1$, and, more importantly, returns a sequence $p_0, \ldots, p_{i-1}$ such that $\rho(A_{p_{i-1}} \ldots A_{p_0}) \approx \hat{\rho}(M_0)$ to any desired level of accuracy. Computing $\Sigma_1 = \Sigma_0 \setminus A_{p_j}$ for one $j \in \{0, \ldots, i-1\}$ and iterating allows one to obtain a Σ_* such that $\hat{\rho}(\Sigma_*) < 1$.

In the adaptive co-simulation of the system introduced in Figure 5.4, we have that $\Sigma_* = \Sigma_0$ excludes the matrix $A_{\text{cs\_2}}$, and $\hat{\rho}(\Sigma_0) \leq 0.992905$.

5.4.3 Conservativeness

As the previous result shows, applying this procedure to the adaptive co-simulation introduced in the previous sub-section results in a stable adaptive co-simulation that will never use the matrix $A_{\text{cs\_2}}$. This is too restrictive. To see why, note that, as illustrated in plots of Figure 5.4, a careful use of the decisions embedded in matrix $A_{\text{cs\_2}}$ actually yields a co-simulation that outperforms the other non-adaptive co-simulations (see the stable trajectory `x1_cs_3` in Table 5.1).

For now, we propose a straightforward solution to this problem: apply the stabilization procedure to $Q = \Sigma^q$, which includes all products of length q of matrices in Σ for a given $q > 0$. The matrix products in the stabilized Q_* may include combinations of matrices that would otherwise have been removed.

Remark 2. There is little use in applying the stabilization to the set of all products of length *up to* q. To see why, suppose we have an unstable product $A_{\text{len n}}$ of length $n < q$. Such product $A_{\text{len n}}$ is the prefix of multiple products of length q. These products of length q represent all possible policies that end with product $A_{\text{len n}}$, so the stabilization of the former set of products might still allow for some products of which $A_{\text{len n}}$ is a prefix.

In the adaptive co-simulation example, we set $q = 2$ and we obtain Q_* that only excludes the matrix $A_{\text{cs\_2}}A_{\text{cs\_2}}$, which means that the policies embedded in $A_{\text{cs\_2}}$ can still be used, provided that they are alternated with any other policies. Applying the algorithm in [233] yields $\hat{\rho}(\Sigma_0^2) \leq 0.982986$.

The next section discusses the implementation of the resulting stabilized master algorithm.

5.4.4 Implementation

If the stabilization procedure terminates, we are left with Q_*: a set of sequences of matrix products of length q. Since we abstracted how each decision is taken at each co-simulation step, we still need to ensure that, at run-time, the decisions taken by the adaptive co-simulation remain within the allowed decisions (in the set Q_*).

To shed light on this problem, note that each sequence $p_0, \ldots, p_{q-1}$ that induces the matrix product $A_{p_{q-1}} \ldots A_{p_0} \in \Sigma^q$, can be associated with one, and only one, natural number $d_{p_0 \ldots p_{q-1}} \in \mathbb{N}_0$ computed as a conversion from base-m digit to a decimal number:

$$d_{p_0 \ldots p_{q-1}} = \sum_{j=0}^{q-1} p_j \cdot m^j, \tag{5.17}$$

where m is the number of matrices in Σ.

We therefore propose to allocate a m^q-bit array, where the position $d_{p_0 \ldots p_{q-1}}$ of the array indicates whether the matrix product $A_{p_{q-1}} \ldots A_{p_0} \in Q_*$. Then, at time t_i with $i \geq q-1$, the previous q policies are used to reconstruct $d_{\sigma_{i-q-1} \ldots \sigma_i}$ and check whether the corresponding sub-sequence is safe to take. If the policy $\sigma(i)$ is not safe to take, then the immediate neighbors of $d_{\sigma_{i-q-1} \ldots \sigma_i}$ in the bit array can be inspected to find whether there are safe policies that can be selected. Figure 5.6 illustrates a scenario where $q = 3, m = 16$ and the master is about to decide to use the policies embedded in matrix A_{14}. A quick look-up to position 2158, obtained with Equation (5.17), shows that this is not allowed. The neighboring positions show alternative matrices that can be used.

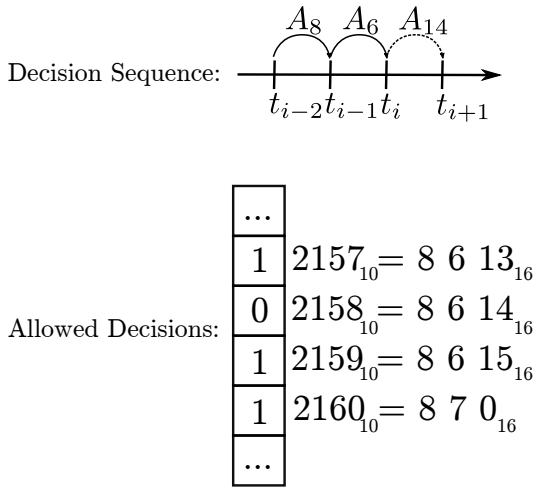


Figure 5.6: Runtime structures of decision sequence monitor. Matrices A_8, A_6, A_{14} are arbitrary matrices.

5.5 Minimizing Forbidden Sequences

The previous section described a straightforward approach to minimize the cost of opportunity of forbidding unstable sequences. In this section, we focus on this problem, while covering a broader spectrum of adaptive master algorithms: those whose set of past policies can influence the future policies.

5.5.1 Constrained Switched Systems

In practice, some switching signals of System (5.14) may not be relevant, and a way to represent the sensible ones is required. For instance, in the switched system described in Example 19, it may not make sense that the Runge-Kutta method is used right after a Forward Euler. This is because the convergence rate of each method is too different to warrant a switch without first taking a step with the Midpoint method.

To model these constraints, we introduce the notion of constrained switched system. When compared to System (5.14), constrained switched systems incorporate a representation of the allowed switching signals using an automaton [240].

Definition 16. Given a bounded set of matrices $\mathcal{A} = \{A_1, \ldots, A_m\}$, we define an *automaton* as a directed and labelled graph $\mathbf{G} = (V, E)$, with nodes V and edges E such that no node has zero ingoing or outgoing degree. Each edge $(v, w, \sigma) \in E$ represents a transition from node $v \in V$ to node $w \in V$, where $\sigma \in \{1, \ldots, m\}$ is the *label*, corresponding to A_σ.

An example automaton, illustrating possible constraints on the system described in Example 19, is shown in Figure 5.7.

We say that the switching signal, or word, $s = \sigma_0\sigma_1 \ldots \sigma_{k-1}$ is *accepted* by an automaton $\mathbf{G}$ if it corresponds to a path in $\mathbf{G}$, that is, if there exists $v_0, v_1, \ldots, v_k \in V$, such that $(v_j, v_{j+1}, \sigma_j) \in E$ for all $j = 0, \ldots, k-1$. An accepted word induces an accepted matrix product $A_s = A_{\sigma_{k-1}} \cdots A_{\sigma_1} A_{\sigma_0} \in \mathcal{A}^k$.

For any $k > 0$, the word s^k is the concatenation of s with itself $k-1$ times.

For example, the word $(fe, 0.001), (fe, 0.002), (md, 0.002)$ is accepted by the automaton shown in Figure 5.7. This word induces the matrix product $\tilde{A}_{md,0.002}\tilde{A}_{fe,0.002}\tilde{A}_{fe,0.001}$.

We denote the set of accepted words of length k as $\mathbf{G}_k$, and the set of all words accepted by the automaton as $\mathbf{G}^* = \bigcup_{k=1}^{\infty} \mathbf{G}_k$. Moreover, $\mathbf{G}_k^\circ$ denotes the set of accepted cycles of length k.

For example,

$$(fe, 0.001), (fe, 0.002), (md, 0.002) \in \mathbf{G}_3, \text{ and}$$
$$(fe, 0.002), (fe, 0.001) \in \mathbf{G}_2^\circ.$$

One can see that given a word $\sigma(0) \ldots \sigma(k-1) \in \mathbf{G}_k$, any sub-word $\sigma(i) \ldots \sigma(j)$ for any $0 \leq i \leq j < k$, satisfies $\sigma(i) \ldots \sigma(j) \in \mathbf{G}_{j-i+1}$. Moreover, since every node has at least one outgoing edge in Definition 16, for any $k' > k$, there exists $\sigma(k) \ldots \sigma(k'-1)$ such that $\sigma(0) \ldots \sigma(k'-1) \in \mathbf{G}_{k'}$.

Definition 17 (CSS)**.** Given a set of matrices $\mathcal{A} = \{A_1, \ldots, A_m\}$, and an automaton $\mathbf{G} = (V, E)$, we define a *constrained switched system* (CSS) $S = \langle \mathcal{A}, \mathbf{G} \rangle$ as a system

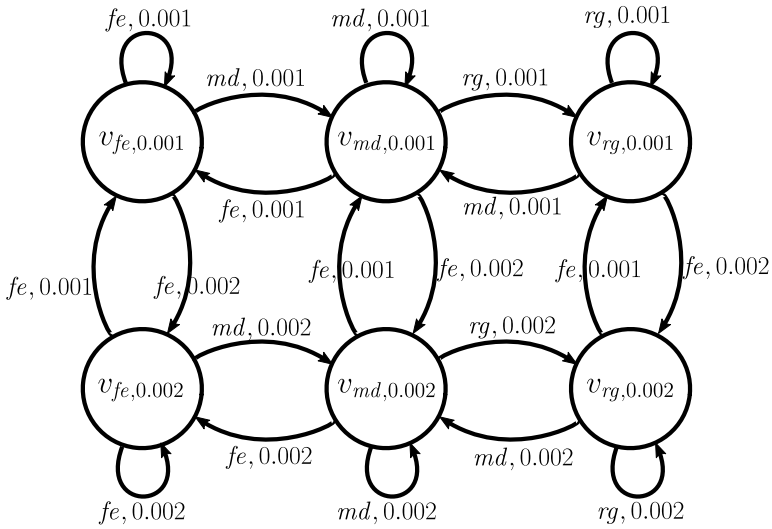


Figure 5.7: Example automaton for Example 19.

where the variable x_k satisfies:

$$x_{k+1} = A_{\sigma_k} x_k : \quad \sigma_0 \dots \sigma_{k-1} \in \mathbf{G}_k. \tag{5.18}$$

We say that System (5.18) is *stable* iff

$$\lim_{k\to\infty} \|x_k\| = \lim_{k\to\infty} \left\| A_{\sigma_{k-1}} \cdots A_{\sigma_0} x_0 \right\| = 0,$$

for any word $\sigma_0 \dots \sigma_{k-1} \in \mathbf{G}_k$ and any $x_0 \in \mathbb{R}^n$.

To determine the stability of a CSS, we introduce the *constrained joint spectral radius*.

Definition 18 ([101, Definition 1.2])**.** The *constrained joint spectral radius* is defined as

$$\hat{\rho}(S) = \lim_{k\to\infty} \hat{\rho}_k(S) \text{ where } \hat{\rho}_k(S) = \sup_{w\in\mathbf{G}_k} \|A_w\|^{\frac{1}{k}},$$

and $\|\cdot\|$ is any matrix norm that satisfies the sub-multiplicative property.

Proposition 1 ([101, Lemma 3.1])**.** *If $\hat{\rho}(S) < 1$, then the CSS is stable.*

It can be shown [197, Lemma 1.2] that $\lim_{k\to\infty} \hat{\rho}_k(S) = \inf_{k\geq 1} \hat{\rho}_k(S)$. Therefore, for any $k > 0$, $\hat{\rho}_k(S)$ is an upper bound to $\hat{\rho}(S)$. This fact, together with Proposition 1, gives us a way to check whether a given CSS S is stable:

1. Pick a finite $k > 0$, and compute $\hat{\rho}_k(S)$;
2. If $\hat{\rho}_k(S) < 1$, then $\hat{\rho}(S) < 1$ and S is stable;
3. Otherwise, pick a larger k and try again.

If the CSS is unstable, then the above procedure will never terminate.

A way to prove that a CSS is unstable is to find a switching signal that causes the system to be unstable. For example, by finding a cycle $c \in \mathbf{G}_k^\circ$ with $\rho(A_c) \geq 1$, where $\rho(A_c)$ denotes

the spectral radius of the matrix A_c induced by the cycle. In other words, finding a matrix product that, when repeated forever, causes the system to be unstable.

Unstable cycles can be found by brute force or branch-and-bound variants [164, 167, 198]. Naturally, these methods look first for unstable cycles with a small length. However, finding longer cycles becomes prohibitively high (see Section 5.10).

For example, a simple (and naive) procedure to find a cycle is to pick a finite k, enumerate all cycles $c \in \mathbf{G}_k^\circ$, check whether $\rho(A_c) \geq 1$, and stop when one such cycle is found.

The method introduced in [234] works well for finding long cycles. To certify the stability of a given CSS, it solves a semidefinite program to compute polynomial Lyapunov functions of degree $2d$. If the program is infeasible, it uses the dual certificate of infeasibity to generate an infinite switching signal of guaranteed growth rate. Subwords of this signal can be used to find unstable cycles. As cycles are found along an infinite switching signal, finding *long* unstable cycles is not particularly more difficult. Moreover, if no unstable cycle can be found, one can retry with polynomial Lyapunov functions of degree $2(d+1)$. There is an improved guarantee on the growth rate of the infinite switching signal as the degree increases.

The following definition formalizes the spectral radius of cycle induced matrix products.

Definition 19 ([101, Definition 1.2])**.** The *generalized spectral radius* of a CSS S is defined as:

$$\rho(S) = \limsup_{k\to\infty} \rho_k(S) \text{ where } \rho_k(S) = \sup_{c\in\mathbf{G}_k^\circ} \rho(A_c)^{\frac{1}{k}} \tag{5.19}$$

It follows [197, Proposition 1.6] that, for finite $k > 0$,

$$\rho_k(S) \leq \rho(S) \leq \hat{\rho}(S) \leq \hat{\rho}_k(S).$$

Moreover, since $\mathcal{A}$ is bounded, it is shown in [101, Theorem A] that $\rho(S) = \hat{\rho}(S)$.

Remark 3. The above discussion about proving that a CSS is unstable focused on finding finite cycles, as opposed to infinite paths. In fact, there is no guarantee that if a CSS satisfies $\hat{\rho}(S) \geq 1$ (i.e., is unstable), then a cycle c with finite length exists, with $\rho(A_c) \geq 1$ (see [197, Section 2.4] and [49, Theorem 2]). However, the systems we experimented with, either satisfy $\hat{\rho}(S) > 1$, or $\hat{\rho}(S) < 1$. For these, the following result was used.

Proposition 2 ([197, Theorem 2.3])**.** *If $\hat{\rho}(S) > 1$, then there exists a cycle $c \in \mathbf{G}_k^\circ$ of length k that satisfies $\rho(A_c) \geq 1$.*

Our goal is to optimally modify a given CSS, by forbidding unstable switching signal cycles from the language it generates. The problem of finding such cycles is outside the scope of our work (see [233] for the algorithm we used, and references thereof for algorithms with the same goal). As such, we introduce the following definition, which represents any algorithm available for this purpose.

Definition 20 (Oracle)**.** Given $\epsilon > 0$, we define a stability oracle $\mathcal{O}_\epsilon : S \to \{\texttt{Stable}\} \cup \bigcup_{k=1}^{\infty} \mathbf{G}_k^\circ$, where S is a CSS. The oracle $\mathcal{O}_\epsilon$ returns either `Stable` certifying that $\hat{\rho}(S) < 1$ or a cycle $c \in \mathbf{G}_k^\circ$ such that $\rho(A_c)^{1/k} > 1 - \epsilon$.

We emphasize that the oracle has a (slightly) imperfect behaviour: in case $1-\epsilon < \hat{\rho}(S) < 1$, one cannot guarantee what the outcome of the oracle will be. This imperfection is intentional

(see Remark 3), as it models the state of the art [292]. Proposition 2 ensures that if $\hat{\rho}(S) > 1 - \epsilon$, there exists a k and a cycle $c \in \mathbf{G}_k^\circ$ such that $\rho(A_c)^{1/k} > 1 - \epsilon$.

We now proceed to define the set of possible different switching signals that are admissible.

Definition 21 (Admissible Regular Language)**.** We say that $\mathcal{L} = \mathbf{G}^*$ is the language *recognized* by the automaton $\mathbf{G}$. A language is *regular* if it is recognized by a finite automaton. A language $\mathcal{L}$ recognized by an automaton $\mathbf{G}$ is *admissible* for $\mathcal{A}$ if the constrained switched system $S = \langle \mathcal{A}, \mathbf{G} \rangle$ satisfies $\hat{\rho}(S) < 1$.

Let $\mathcal{L}_0$ denote the language recognized by the automaton $\mathbf{G}_0$ of a given $S = \langle \mathcal{A}, \mathbf{G}_0 \rangle$. Informally, our goal is to find the "largest" regular language $\mathcal{L}^\star \subseteq \mathcal{L}_0$ that is admissible. For this optimization problem to be well defined we need to find a metric for the objective. This metric should be in accordance to the fact that given $\mathcal{L} \subseteq \mathcal{L}'$, the objective should favor $\mathcal{L}'$. A widely used notion to describe the size of a regular language is that of Entropy.

Definition 22 (Entropy [239, Definition 4.1.1])**.** Given a regular language $\mathcal{L}$ recognized by an automaton $\mathbf{G}$, we define the *entropy* as

$$h(\mathcal{L}) = \lim_{k \to \infty} \frac{1}{k} \log_2 |\mathbf{G}_k|.$$

In the above, $|\mathbf{G}_k|$ represents the number of words of length k accepted by the automaton $\mathbf{G}$.

We denote the entropy of the language $\mathbf{G}^*$ recognized by an automaton $\mathbf{G}$ as $h^*(\mathbf{G})$.

If $\mathcal{L} \subseteq \mathcal{L}'$, then $\mathbf{G}_k \subseteq \mathbf{G}'_k$ for any k, and so $h(\mathcal{L}) \leq h(\mathcal{L}')$. Our problem can now be formulated.

Problem 1. *Given a CSS $\langle \mathcal{A}, \mathbf{G}_0 \rangle$, find the language $\mathcal{L}^\star$ solution of the following optimization problem:*

$$\begin{aligned} \mathcal{L}^\star = \sup_{\mathcal{L} \text{ regular}} \; & h(\mathcal{L}) \text{ s.t.} \\ & \mathcal{L} \subseteq \mathcal{L}_0, \\ & \mathcal{L} \text{ is admissible for } \mathcal{A}. \end{aligned} \tag{5.20}$$

where $\mathcal{L}_0$ is the language recognized by $\mathbf{G}_0$.

Remark 4. In Problem 1, we restrict our attention to regular languages. While there are examples that highlight the benefit of using non-regular languages (see Example 20), in practice, one needs an *efficient* way of generating accepted switching signals. For instance, during a co-simulation, at any step, the simulators need to compute as quickly as possible the set of policies that can be taken (see [155, Section 4.4] for how this can be done). Automata allow the decision procedure to be fast, with little memory. In addition, as hinted in Example 21, regular languages may be constructed to approximate an admissible language with entropy arbitrarily close to the entropy of the optimal solution, even if that optimal solution is a non-regular language.

Example 20. Consider $\mathcal{A} = \{A_1, A_2\}$, with $A_1 = 2$ and $A_2 = \frac{1}{2}$, and $\mathbf{G} = (V, E)$, where $V = \{v_1\}$ and $E = \{(v_1, v_1, 1), (v_1, v_1, 2)\}$. That is, $\mathbf{G}$ has the form

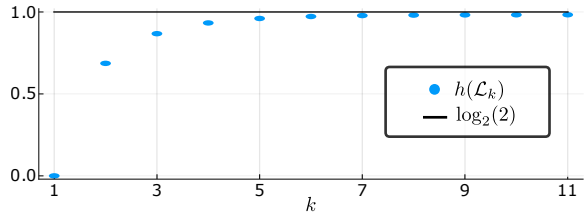


Figure 5.8: Evolution of $h(\mathcal{L}_k)$ of Example 21 in terms of k.

The optimal solution $\mathcal{L}^\star$ of (relaxed) Problem 1 should include every word that has more 1s than 2s, because $A_1^r \cdot A_2^s < 1$ iff $r < s$. As shown in [333, Example 1.73], no automaton can be built that accepts this language.

Example 21. Consider $\mathcal{A} = \{A_1, A_2\}$, with $A_1 = 1$ and $A_2 = \frac{1}{2}$. A language is admissible if it does not contain the infinite repetition of the symbol 1. Let $\mathcal{L}_k$ be language of all words that do not contain k consecutive 1's. Figure 5.8 suggests that $h(\mathcal{L}_k)$ tends to $\log_2(2)$ when k tends to infinity. The quantity $\log_2(2)$ denotes the entropy of the optimal solution.

5.6 Lift-and-Constrain Stabilization

5.6.1 Constraining for more stability

Algorithm 5 details an iterative procedure that stabilizes a given CSS $S = \langle \mathcal{A}, \mathbf{G} \rangle$, using the oracle in Definition 20. At each iteration, if the oracle returns a cycle $c = \sigma_k \dots \sigma_k$, then c is eliminated from $\mathbf{G}$. The removal of a cycle can be accomplished by removing an edge of $\mathbf{G}$, thus potentially decreasing $\hat{\rho}(S)$. After removing the cycle c, any infinite sequence in $\mathbf{G}^*$ for which c is a subsequence will be eliminated too. This is illustrated in Example 22. The algorithm can produce an empty CSS, which does not imply that the original CSS is impossible to stabilize. An empty CSS is trivially stable.

Example 22. Consider the automaton in Figure 5.9, and suppose the oracle has returned the cycle 234. This cycle is highlighted in red, in the figure. Any of the edges in red can be removed to forbid the unstable sequence. If edge $v_1 \xrightarrow{2} v_2$ is removed, the infinite sequences accepted by the resulting automaton end with either an infinite sequence of 2's, or an infinite sequence of 3's. If edge $v_2 \xrightarrow{3} v_3$ is removed instead, the resulting automaton accepts infinite sequences comprised of repeating subsequences which include 2, or 3, or 12.

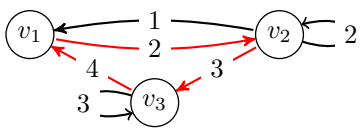


Figure 5.9: Automaton of Example 22.

As Example 22 shows, the choice of different edges to be removed has a different impact in the entropy of the resulting automaton. Informally, removing the edge $v_2 \xrightarrow{3} v_3$ seems

to be the best choice because the resulting automata allows for *more* sequences. This is corroborated by computing the entropy of the resulting automaton alternatives. See Section 5.8 for how to compute the entropy in this example.

ALGORITHM 5: Stabilization algorithm for a constrained switched system. $h^*(\mathbf{G})$ denotes the entropy of the language recognized by $\mathbf{G}$. The difference $\mathbf{G} - e$ denotes the automaton obtained by removing the edge e from $\mathbf{G}$.

Data: A CSS $S = \langle \mathcal{A}, \mathbf{G} \rangle$.
Result: A stable CSS $S = \langle \mathcal{A}, \mathbf{G} \rangle$.
1 **while** $\mathcal{O}_\epsilon(S) \neq$ `Stable` **do**
  1. Find $e \in \arg\max\{\, h^*(\mathbf{G} - e) \mid e \in E, e$ is an edge of the cycle $\mathcal{O}_\epsilon(S)\,\}$;
  2. Set $\mathbf{G} := \mathbf{G} - e$;

2 **end**

The following result demonstrates that Algorithm 5 always terminates.
Theorem 1. *Given a CSS $S = \langle \mathcal{A}, \mathbf{G} \rangle$ and an oracle satisfying Definition 20, Algorithm 5 terminates in finite time and the resulting CSS is stable.*

Proof. At each iteration of the algorithm, the number of edges of the automaton $\mathbf{G} = (V, E)$ decreases by one. Since at the beginning of the algorithm $|E|$ is finite, the algorithm must terminate after a finite number of iterations. The condition for termination of Algorithm 5 implies that the resulting system is stable. □

Remark 5. In Theorem 1, the assumption that the oracle in Definition 20 always terminates is crucial, as the problem solved by the oracle is undecidable in general (recall Remark 3).

5.6.2 Lifting for less conservativeness

Algorithm 5 takes a constrained switched system $S = \langle \mathcal{A}, \mathbf{G} \rangle$, and outputs a constrained switched system $S' = \langle \mathcal{A}', \mathbf{G}' \rangle$ that is stable, while attempting to maximize the entropy of the language recognized by $\mathbf{G}'$. If we let $\mathcal{L}'$ denote this language, then, relating this to Problem 1, $\mathcal{L}'$ is admissible and regular, and thereby a potential solution. However, it may not be the optimal solution. Similarly, if the algorithm returns an empty CSS, this does not mean that the original CSS is impossible to stabilize, as illustrated in Example 23. To maximize the entropy of the stabilized CSS's, we propose to take an *M-Path-Dependent lift* of the automaton representing the input language $\mathcal{L}_0$.
Example 23. Consider the two graphs depicted in Figure 5.10, and assume that the $\rho(A_1) > 1$, $\rho(A_2) > 1$, and $\rho(A_1 \cdot A_2) > 1$ Our stabilization procedure, when applied to the graph on the left will remove the two edges and return an empty CSS, but when applied to the graph on the right, will return a non-empty CSS. The edges removed are highlighted in red.
Definition 23 ([300, Definition 3])**.** Given an automaton $\mathbf{G}$, we define the *lifted* automaton $\mathbf{G}^{[k]}$ of degree k as follows. For each path $v_0, \sigma_0, v_1, \sigma_1, \ldots, \sigma_k, v_{k+1}$ with length $k+1$ of $\mathbf{G}$, $\mathbf{G}^{[k]}$ has a node $u^- = v_0\sigma_0 v_1 \sigma_1 \ldots \sigma_{k-1} v_k$, a node $u^+ = v_1\sigma_1 v_2 \sigma_2 \ldots \sigma_k v_{k+1}$ and an edge (u^-, u^+, σ_k).

Figure 5.11 shows the second degree ($k = 2$) lift of the automaton in Figure 5.9.

The lifted automaton represents the same language, as shown by Proposition 3, but, as suggested by Theorem 2 (proved below) and illustrated by Example 21, lifting the automaton

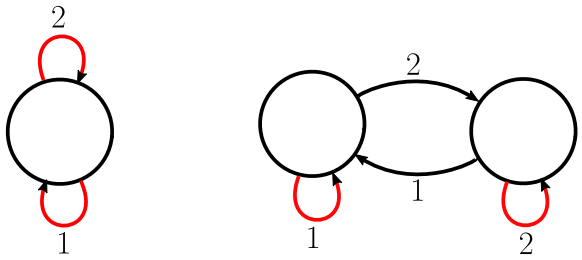


Figure 5.10: Graphs described in Example 23.

before applying Algorithm 5 allows one to obtain an admissible language with higher entropy.

Proposition 3. *Let $\mathcal{L}$ be the language recognized by $\mathbf{G}$ and $\mathcal{L}^{[k]}$ the language recognized by $\mathbf{G}^{[k]}$, where $\mathbf{G}^{[k]}$ is the lift of degree k of $\mathbf{G}$. Then $\mathcal{L} = \mathcal{L}^{[k]}$.*

Proof. Consider a sequence $\sigma_0 \dots \sigma_{i-1}$.

If $\sigma_0 \dots \sigma_{i-1} \in \mathbf{G}_i$, there exists nodes $v_0, v_1, \dots, v_i$ of $\mathbf{G}$ such that

$$v_0, \sigma_0, v_1, \sigma_1, \dots, \sigma_{i-1}, v_i$$

is a path of $\mathbf{G}$. As no node has zero ingoing degree, there exists a path of length k that ends in node v_0, denoted as $v_{-k}, \sigma_{-k}, \dots, \sigma_{-1}, v_0$ in $\mathbf{G}$. By Definition 23, for any $j = 0, \dots, i$, $u_j = v_{j-k}\sigma_{j-k} \dots \sigma_{j-1}v_j$ is a node of $\mathbf{G}^{[k]}$ and for any $j = 0, \dots, i-1$, there is an edge (u_j, u_{j+1}, σ_j) in $\mathbf{G}^{[k]}$. Therefore $\sigma_0 \dots \sigma_{i-1} \in \mathbf{G}_i^{[k]}$.

If $\sigma_0 \dots \sigma_{i-1} \in \mathbf{G}_i^{[k]}$, there exists nodes $u_0, u_1, \dots, u_i$ of $\mathbf{G}^{[k]}$ such that $u_0, \sigma_0, u_1, \sigma_1, \dots, \sigma_{i-1}, u_i$ is a path of $\mathbf{G}^{[k]}$. Let $v_{-k}, \dots, v_i$ be the nodes of $\mathbf{G}$ and $\sigma_{-k}, \dots, \sigma_{-1}$ be the symbols such that for any $j = 0, \dots, i$, $u_j = v_{j-k}\sigma_{j-k} \dots \sigma_{j-1}v_j$. By Definition 23, $v_0, \sigma_0, v_1, \sigma_1, \dots, \sigma_{i-1}, v_i$ is a path of $\mathbf{G}$ hence $\sigma_0 \dots \sigma_{i-1} \in \mathbf{G}_i$.

□

The following theorem suggests that, at least for a single cycle, it is always better to stabilize the lifted graph.

Theorem 2. *Consider Algorithm 5 with input $\mathcal{A}, \mathbf{G}_0^{[k]}$ (resp. $\mathcal{A}, \mathbf{G}_0^{[k+1]}$) where $\mathbf{G}_0^{[k]}$ (resp. $\mathbf{G}_0^{[k+1]}$) is the lift of degree k (resp. $k+1$) of a given automaton $\mathbf{G}$. If $\mathcal{O}_\epsilon(\mathcal{A}, \mathbf{G}_0^{[k]})$ and $\mathcal{O}_\epsilon(\mathcal{A}, \mathbf{G}_0^{[k+1]})$ are cycles corresponding to the same word, then $h^*(\mathbf{G}_1^{[k]}) \leq h^*(\mathbf{G}_1^{[k+1]})$.*

Proof. Let e be the edge such that $\mathbf{G}_1^{[k]} = \mathbf{G}_0^{[k]} - e$, that is, the edge removed by the algorithm for $\mathbf{G}_0^{[k]}$. Let $\sigma_1\sigma_2 \dots \sigma_k\sigma_{k+1}\sigma_{k+2}$ be a sub-word of the repetition of the cycle c and $v_1, v_2, \dots, v_{k+3}$ be such that

$$e = (v_1\sigma_1v_2\sigma_2 \dots \sigma_k v_{k+1}, v_2\sigma_2 \dots \sigma_k v_{k+1}\sigma_{k+1}v_{k+2}, \sigma_{k+1})$$

and $(v_{k+2}, v_{k+3}, \sigma_{k+2})$ is an edge of $\mathbf{G}$. Let $\mathbf{G}_0^{[k+1]'}$ be the graph obtained by removing the node $v_1\sigma_1v_2\sigma_2 \dots \sigma_{k+1}v_{k+2}$ in $\mathbf{G}_0^{[k+1]}$. The two automata $\mathbf{G}_1^{[k]}$ and $\mathbf{G}_0^{[k+1]'}$ recognize the same language. Let $\mathbf{G}_0^{[k+1]''}$ be the graph obtained by removing the edge

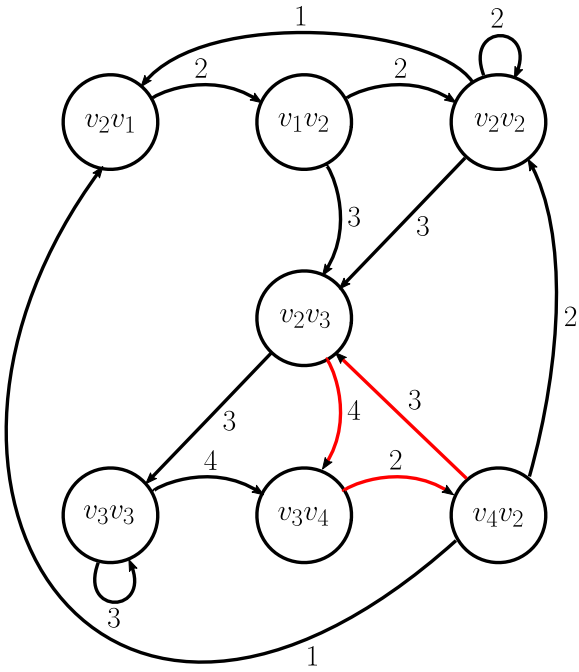


Figure 5.11: Second degree lifted automaton of Example 22.

$e' = (v_1\sigma_1 v_2\sigma_2 \dots \sigma_{k+1}v_{k+2}, v_2\sigma_2 v_3\sigma_3 \dots \sigma_{k+2}v_{k+3}, \sigma_{k+2})$ in $\mathbf{G}_0^{[k+1]}$. The language recognized by $\mathbf{G}_0^{[k+1]'}$ is a subset of the language recognized by $\mathbf{G}_0^{[k+1]''}$.

Moreover, as e' is an edge of the cycle $\mathcal{O}_\epsilon(\mathcal{A}, \mathbf{G}_0^{[k+1]})$, $h^*(\mathbf{G}_0^{[k+1]''}) \leq h^*(\mathbf{G}_1^{[k+1]})$. Therefore

$$h^*(\mathbf{G}^{[k]}{}_0) = h^*(\mathbf{G}_0^{[k+1]'}) \leq h^*(\mathbf{G}_0^{[k+1]''}) \leq h^*(\mathbf{G}_1^{[k+1]}).$$

□

In Section 5.10 we show results corroborating Theorem 2.

5.7 Implementation

The implementation of the stabilization of a CSS is summarized as follows:

1. find all unstable cycles of length up to 3 using brute force enumeration;
2. since several cycles can be disallowed by removing a single edge, select the edge that disallows the largest number of unstable cycles, and use the entropy of the resulting graph to break ties;
3. repeat steps 1–2 until all allowed cycles have a spectral radius below 1;
4. Use the method of [234] to determine whether the resulting CSS is stable or whether there is an unstable cycle.
5. if there is an unstable cycle, select the edge that maximizes the entropy of the resulting system (steps 1–2 of Algorithm 5);

6. repeat steps 4–5 until the resulting system is stable.

It is easy to see that this implementation is a realization of Algorithm 5. Steps 1–4 are an optimization since they execute relatively quickly, and make the execution of the method in [234] take less time.

In Step 6, instead of computing the entropy, we compute the spectral radius of the adjacency matrix of the resulting system. This is equivalent to maximizing the entropy, as the next section shows.

5.8 Computation of the Entropy

5.8.1 Spectral Radius of Adjacency Matrix

Consider a given CSS $S = \langle \mathcal{A}, \mathbf{G} \rangle$, and let B be the adjacency matrix of $\mathbf{G}$.

The matrix element b_{ij} of B^k gives the number of different paths of length k from node i to node j [374]. Hence, $\left\|B^k\right\|$ gives a measure of the number of different paths from each node to each other node (see, e.g., [233, Remark 2]), and $\left\|B^k\right\|^{\frac{1}{k}}$ gives the growth rate of this quantity. Since we're considering infinitely running co-simulations, taking the limit $k \to \infty$, we have the spectral radius of the adjacency matrix:

$$\rho(B) = \lim_{k \to \infty} \left\|B^k\right\|^{1/k}.$$

Example 24. Recall Example 22, let B_1 denote the adjacency matrix of the automata in Figure 5.9 without the edge $v_1 \xrightarrow{2} v_2$, and let B_2 denote the adjacency matrix of the same automata, without the edge $v_2 \xrightarrow{3} v_3$. Then $\rho(B_1) = 1 < \rho(B_2) \approx 1.6180$.

5.8.2 Edge Shift

The logarithm of the spectral radius of the adjacency matrix of an *irreducible* automaton gives the entropy of its *edge shift* [239, Theorem 4.3.1]. An automaton is *irreducible* if for every pair of nodes u, v, there exists a path from u to v accepted by the automaton. In other words, the graph consists of a single strong component. It turns out that the entropy of the edge shift is equal to the entropy of the language recognized by the automaton if the automaton is *right-resolving* [239, Proposition 4.1.13].

Definition 24 ([239, Definition 2.2.5])**.** The *edge shift* of an automaton $\mathbf{G} = (V, E)$ is the language recognized by the automaton $\mathbf{G}' = (E, E')$ with the transitions

$$((u, v, \sigma), (v, w, \sigma'), (v, w, \sigma')) \in E'$$

for each $(u, v, \sigma), (v, w, \sigma') \in E$.

An edge shift of automaton Figure 5.12a is illustrated in Figure 5.12b.

Definition 25 ([239, Definition 3.3.1])**.** An automaton $\mathbf{G}$ is *right-resolving* if for every vertex v, the outgoing edges have different symbols.

Every regular language is recognized by a right-resolving automaton. Moreover, there are automated ways to obtain such an automaton from a starting representation of a language with an automaton that is not right-resolving [239, Section 3.3].

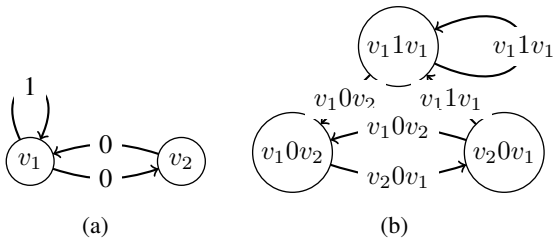


Figure 5.12: An automaton (a) and its edge shift (b).

Since the automata considered here are right resolving and irreducible, by [239, Theorem 4.3.1] and [239, Proposition 4.1.13], the entropy is computed by computing the spectral radius of the adjacency matrix of the CSS.

5.9 Optimality

The solution attained by Algorithm 5 is not necessarily the optimal solution. For once, applying different lift degrees will yield different optimal solutions. Second, Algorithm 5 removes an edge before finding the next unstable cycle, which means that it misses the chance of optimizing which edge to remove, when more cycles are available (recall steps 1–3 of the above implementation).

Unfortunately, we found no way of guessing which lift degree yields the optimal solution. However, with small enough constrained switched systems (in the number of matrices and states), it is possible to find the optimal solution, for a given lift degree k.

To find the optimal solution, suppose that, for a CSS with a lift degree k, we know what the unstable cycles are. Now we can iterate over all possible procedures to disallow these cycles in the CSS (each procedure is a sequence of edges to be removed), and compute the entropy of the resulting CSS. The optimal solution is the one that has the maximal entropy.

In order to collect all the unstable cycles, the following procedure can be used:

1. given a CSS with a lift of degree k, apply Algorithm 5 to find an admissible language, and record all the cycles that were removed throughout the procedure;
2. iterate over all possible ways of disallowing the cycles on the original CSS with a lift of degree k, and apply the one that results in a language with maximal entropy;
3. the resulting language is not necessarily admissible, because the best procedure is not necessarily that same as the one picked by Algorithm 5 in Step 1, so apply Algorithm 5 to identify and disallow the remaining cycles, adding these to the set of unstable cycles.
4. now repeat Steps 2–3, collecting more and more unstable cycles, until the set of unstable cycles does not change.

We stress that the above procedure can only be applied to very simple CSS.

The resulting set of unstable cycles represent all possible unstable cycles, and the admissible

language found is the optimal solution.

An example application is described in Section 5.10. See [197, 331] for other applications of switched systems, where our technique could be applied as well.

5.10 Application

We apply our algorithm to generate a numerically stable adaptive master algorithm for the co-simulation of a controlled inverted pendulum. We will consider two simulators.

As described in Section 5.3.1, in the context of co-simulation, time is discretized into a countable set $T = \{t_0, t_1, t_2, \ldots\} \subset \mathbb{R}$, where $t_{i+1} = t_i + H_i$ is the time at step i and H_i is the communication step size at step i, with $i = 0, 1, \ldots$. From time $t_i \to t_{i+1}$, the simulator S_j, with $j = 1, 2$, is a mapping,

$$\begin{aligned} \tilde{x}_j(t_{i+1}) &= F_j(t_i, \tilde{x}_j(t_i), u_j(t_i)) \\ y_j(t_i) &= G_j(t_i, \tilde{x}_j(t_i), u_j(t_i)) \end{aligned} \tag{5.21}$$

with state vector $\tilde{x}_j$, input vector u_j and output vector y_j.

Simulators exchange outputs only at times $t \in T$. We assume without loss of generality[1] that the two simulators are coupled in a feedback loop, that is,

$$u_1 = y_2 \text{ and } u_2 = y_1. \tag{5.22}$$

In the interval $t \in [t_i, t_{i+1}]$, each simulator S_j approximates the solution to a linear ODE,

$$\begin{aligned} \dot{x}_j &= A_j x_j + B_j u_j \\ y_j &= C_j x_j + D_j u_j \end{aligned} \tag{5.23}$$

where A_j, B_j, C_j, D_j are matrices, and the initial state $x_j(t_i)$ is computed in the most recent co-simulation step. To avoid algebraic loops and keep the exposition short, we assume that either D_1 or D_2 is the zero matrix. Let $D_2 = \mathbf{0}$.

Since the simulators only exchange outputs at times $t_i, t_{i+1} \in T$, the input u_j has to be extrapolated in the interval $[t_i, t_{i+1})$. In the simplest co-simulation strategy[2], this extrapolation is often implemented as a zero-order hold: $\tilde{u}_j(t) = u_j(t_i)$, for $t \in [t_i, t_{i+1})$. Then, Equation (5.23) can be re-written to represent the unforced system being integrated by each simulator:

$$\begin{bmatrix} \dot{x}_j \\ \dot{\tilde{u}}_j \end{bmatrix} = \begin{bmatrix} A_j & B_j \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} x_j \\ \tilde{u}_j \end{bmatrix} \tag{5.24}$$

We can represent the multiple internal integration steps of System (5.24), performed by the simulator S_j in the interval $t \in [t_i, t_{i+1}]$, as

$$\begin{bmatrix} \tilde{x}_j(t_{i+1}) \\ \tilde{u}_j(t_{i+1}) \end{bmatrix} = \tilde{A}_j^{k_j} \begin{bmatrix} \tilde{x}_j(t_i) \\ \tilde{u}_j \end{bmatrix} \tag{5.25}$$

[1] The procedure described here can be easily extended to more complex co-simulation scenarios.

[2] The derivation presented can be applied to more sophisticated input extrapolation techniques, see [71, Equation (9)].

where, e.g., $\tilde{A}_j = \mathbf{I} + h_j \begin{bmatrix} A_j & B_j \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$ for the Forward Euler method, $k_j = (t_{i+1} - t_i)/h_j$ is the number of internal steps, and $0 < h_j \leq H_i$ is the internal fixed step size that divides H_i. Note that this equation implements the mapping in Equation (5.21).

At the beginning of the co-simulation step i, $u_1(t_i) = y_2(t_i)$ and $u_2(t_i) = y_1(t_i)$, as in Equation (5.22). This, together with Equation (5.23), gives,

$$\begin{aligned} u_1(t_i) &= C_2 \tilde{x}_2(t_i) \\ u_2(t_i) &= C_1 \tilde{x}_1(t_i) + D_1 C_2 \tilde{x}_2(t_i). \end{aligned} \tag{5.26}$$

Equations (5.24), (5.25), and (5.26) can be used to represent each co-simulation step by a linear mapping

$$\begin{bmatrix} \tilde{x}_1(t_{i+1}) \\ \tilde{x}_2(t_{i+1}) \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \tilde{A}_1^{k_1} & \mathbf{0} \\ \mathbf{0} & \tilde{A}_2^{k_2} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & C_2 \\ \mathbf{0} & \mathbf{I} \\ C_1 & D_1 C_2 \end{bmatrix}}_{\tilde{A}} \begin{bmatrix} \tilde{x}_1(t_i) \\ \tilde{x}_2(t_i) \end{bmatrix}$$

With the present work we generate a stabilized CSS, that encodes the set of all possible sequences of configurations that make the co-simulation stable, which can then be consulted during the co-simulation, with little computational cost, as described in Section 5.4.4.

Consider the co-simulation of an inverted pendulum that is kept at the equilibrium point using a state feedback controller. Simulator S_1 represents the controller, and simulator S_2 represents the pendulum.

Around the equilibrium point, the pendulum can be approximated as a system of the form of Equation (5.23), with

$$\begin{aligned} A_2 &= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -(I+ml^2)(b/p) & (m^2gl^2)/p & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -(mlb)/p & mgl(M+m)/p & 0 \end{bmatrix} \\ B_2 &= \begin{bmatrix} 0 & (I+ml^2)/p & 0 & ml/p \end{bmatrix}^\top \\ C_2 &= \mathbf{I} \qquad D_2 = \mathbf{0} \end{aligned}$$

and parameters $M = 0.5, m = 0.2, b = 0.1, I = 0.006, g = 9.8, l = 0.3$.

The controller is a linear quadratic regulator, which, put in the form of Equation (5.23), is

$$\begin{aligned} A_1 &= \mathbf{0} \qquad B_1 = \mathbf{0} \\ C_1 &= \mathbf{0} \qquad D_1 = \begin{bmatrix} 1.0000 & 1.6567 & -18.6854 & -3.4594 \end{bmatrix}. \end{aligned}$$

Assume we can use the Forward Euler and Midpoint methods, with internal fixed step sizes in the set $\{0.01, 0.02, 0.1, 0.2\}$. Furthermore, the co-simulation step size can be $H = 0.1$ or $H = 0.2$. Note that the internal step sizes must always divide the communication step size H, and that the numerical method and step size used in the controller simulator have no impact in the co-simulation stability, because it has no internal dynamics. Then, applying Equations (5.24), (5.25), and (5.26), we get a switched system over 8 matrices.

The matrices $\boldsymbol{A}_2$ (corresponding to $H = 0.2$, $h_1 = 0.2$, Forward Euler), $\boldsymbol{A}_3$ (corresponding to $H = 0.2$, $h_1 = 0.02$, Midpoint) and $\boldsymbol{A}_4$ (corresponding to $H = 0.2$, $h_1 = 0.02$, Forward Euler) have a spectral radius larger than one. This means that the switching signals $222...$, $333...$ and $444...$ should be forbidden.

Applying Algorithm 5 directly to the unconstrained switched system (which corresponds to a lift of degree 0), leads to removal of the edges with labels $2, 3$ and 4. This completely disallows the use of the matrices $\boldsymbol{A}_2$, $\boldsymbol{A}_3$ and $\boldsymbol{A}_4$. The resulting language turns out to be admissible, its entropy is $\log_2(5)$.

Applying Algorithm 5 to a lift of degree 1, we get a constrained switched system with the automaton shown in Figure 5.13, where the edges in red were removed by the algorithm. We can see that the matrices $\boldsymbol{A}_2$, $\boldsymbol{A}_3$ and $\boldsymbol{A}_4$ are now allowed by the algorithm (only the cyclic application of each one of these matrices is still disallowed). This solution is less conservative than the one with degree 0. Its entropy is $log_2(7.26)$. One allowed cycle is 32645, where the symbols 5 and 6 seem to play the role of stabilizing the cycle.

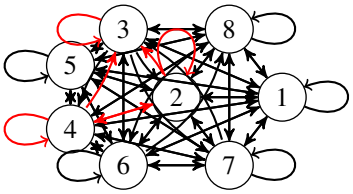


Figure 5.13: Solution with entropy $\log_2(7.2568898)$.

We applied Algorithm 5 to the lifts of degree 0, 1 and 2. At each application of the algorithm, a stable constrained switched system was produced, with an entropy that increased with the degree of the lift. These results, summarized in Table 5.2, corroborate Theorem 2.

Table 5.2: Entropy achieved per lift degree.

k	Entropy [bit]	CPU time [s]
0	$\log_2(5)$	0.13
1	$\log_2(7.2568898)$	1.8
2	$\log_2(7.7083039)$	280

This application to co-simulation illustrates an important advantage of the method presented in [234]: it is capable of finding large unstable cycles. This method does not find the unstable cycles by iterating through the cycles of some length K but instead extracts them from an infinite switching signal, hence it is not harder for the method to find large unstable cycles. For the lift of degree 2 for example, it found the unstable cycle 542245332 of length

9, and in a subsequent iteration found the cycle $22453354224533542245335422453254$ of length 33. A brute force method would have to enumerate all 8^{33} cycles to achieve the stabilization of the adaptive solver.

Regarding the optimality of the solution found for the lift with degree 1, we have applied the procedure detailed in Section 5.9 to confirm that $\log_2(7.2568898)$ is indeed the maximal entropy for that degree.

5.11 Related Work

There has been a huge effort to understand how to ensure that a (discrete time) switched system is stable. We refer the interested reader to [16, 197, 238, 292, 342] for introductions and surveys on this subject. To the best of our knowledge, the problem we introduce here, i.e. finding the largest set of switching signals that guarantees the stability of the system, has never been studied. In the broader field of stabilization of switched systems, we can highlight the works in [168, 178, 221, 299, 306, 379, 380, 389, 390]. The key difference with our work is the goal: we are not satisfied with a single stable switching signal; we want to provide the maximum flexibility to the stabilized CSS, which can make use of this flexibility to choose the most appropriate switching signal. The works in [178, 221, 299, 306, 379, 380] are focused on continuous time systems, and [379, 380, 390] aim at deriving state feedback laws (in addition to switching signals) that make the system stable.

The approach followed in [390] assumes that each mode of the system is stable. In our case, the goal is the same but we tolerate unstable modes.

The approach in [168] is interesting because it allows the combination of stable and unstable modes, in order to ensure stability. However, no algorithm is provided to find these combinations.

The aim of [221] is different as it describes the search for one particular stable trajectory while we maximize the size of a language of stable switching signals.

[334] describes the stability analysis for continuous switched systems with parametric uncertainties.

[389] focuses on proving that a system is stabilizable, rather than making the system stable. It deals with forced discrete time switched systems, and the stabilization procedure finds a control policy (switching signal, and input) that stabilizes the system. This is in contrast to our goal, which is to find all policies that make the system stable, and maximize this set. In the context of co-simulation, the reader can find stability analysis of traditional master algorithms in [71, 153, 325, 326].

5.12 Concluding Remarks

We introduce a new problem in the context of constrained switched systems: 1) to restrict the switching possibilities of the system, so as to ensure its stability, and 2) to leave as many switching policies as possible (provided that the system becomes stable).

The motivation for leaving as many switching policies as possible lies in the fact that, in adaptive co-simulation, the master algorithm will make the best possible choice as a function of information obtained during the simulation. We restrict the switching possibilities to be representable by an automaton because of their great efficiency.

The problem is interesting in that it transforms a control problem into the problem of building an optimal language, that is, optimizing the construction of an automaton. By combining classical control concepts for switched systems (like the CJSR) , with classical automata-theoretic concepts (like the entropy of shifts), one can design algorithms to solve this problem. Our algorithm takes the form of a hierarchy of sufficient conditions, where increasingly better solutions are found by lifting the automaton (see Figure 5.8 and Table 5.2). Essentially, this allows one to control the optimality of the solution, at the cost of processing power and memory.

Limitations and Assumptions This work is limited to LTI systems. A generalization to non-linear systems could be done via piece-wise affine approximations.

This work is aimed to be a proof of concept, and we leave many research questions open. We plan to investigate the conservativeness of restricting ourselves to regular languages (see Example 20). Second, we want to understand how our method can be optimized for the particularizes of co-simulation, and apply it to nonlinear systems. Finally, we plan to modify Algorithm 5 so that stronger theoretical results can be proven.

Chapter 6

Stability Preservation in Hybrid Co-simulation

Disclaimer The content in this chapter is adapted from:

- GOMES, CLÁUDIO, Paschalis Karalis, Eva M. Navarro-López, and Hans Vangheluwe. "Approximated Stability Analysis of Bi-Modal Hybrid Co-Simulation Scenarios." In 1st Workshop on Formal Co-Simulation of Cyber-Physical Systems, 345–60. Trento, Italy: Springer, Cham, 2017. https://doi.org/10.1007/978-3-319-74781-1_24.
- Ongoing work to prepare a manuscript for a conference submission.

In the previous chapter, we discussed how stability, a property that is often satisfied by continuous physical systems, can be preserved under an adaptive co-simulation scheme. In this chapter, we focus on the preservation of the same property, for a restricted class of hybrid systems.

6.1 Introduction

As exemplified in Section 3.8, in hybrid systems, one distinguishes two types of dynamics: the continuous dynamics (usually dictated by a differential equation), and the event-discrete-related dynamics (usually dictated by a finite state machine or a transition system). As a result, in order to make the computation of the combined behavior possible, one needs to not only approximate the behavior of the continuous dynamics, but also to *accurately* and *efficiently* detect when to compute the discrete-event dynamics. In order to accurately do so, transition (or event) detection and location[1] schemes [388] are employed. The efficiency requirement is satisfied by setting the appropriate parameters of the transition location scheme: wrong tolerance values can lead to unnecessary computations or inaccurate results (see Section 6.2).

In real time simulation of hybrid systems, the efficiency requirement is stricter and often only the transition detection is performed, skipping the event location part; or only a *fixed* number of iterations to locate the transition are allowed [87, Section 10.8].

[1] Event location means the search for the exact time at which an event occurs.

Theoretically, any trajectory computed in a co-simulation should be the same as a solution to the original system. In practice, just as with the simulation of systems described by differential equations, this is not the case. As exemplified in Section 6.2, it can happen that an inappropriate transition location scheme causes the co-simulation to fail to preserve the stability of the original system.

Since the simulators have widely different capabilities the master algorithm often has to assume the largest common denominator when coordinating a co-simulation (as happens in the FMI Standard). This means that often there is no ability to rollback or accurately capture state events that impact multiple simulators, making the task of choosing an appropriate step size not trivial (see, e.g., [24] for how to do this with smooth systems), because it directly impacts the transition detection abilities.

It is therefore of utmost importance that master algorithms can tweak the communication frequency of the tools to ensure that system developers can trust the co-simulation results. However, more frequent communication entails a performance toll. Hence, a valid *research question* is: for a particular co-simulation scenario, what is the lowest frequency for which tools can exchange values, that still ensures that the computed trajectories are bounded? The question is not new: it has been studied for traditional simulation, but not for hybrid system co-simulation. We make use of existing stability results for hybrid systems.

Stability of hybrid systems has been studied extensively (see, e.g., [62, 149, 195, 197, 236, 241, 272, 273, 387]). Most of the results define stability in the Lyapunov sense [207] (bounded trajectories, adapted from the continuous smooth case), and can be classified as: (i) the study of stability by using a common energy function for all the subsystems [236], or (ii) the use of multiple Lyapunov functions, one for each subsystem [62, 203, 272]. The consideration of multiple equilibria is not common in the hybrid systems literature, being typically focused on the study of systems with a unique equilibrium point for all the subsystems. Among very few results considering multiple equilibria are [272, 273].

To the best of our knowledge, there is no work that applies these stability analysis techniques to study the effects of co-simulation in hybrid systems. The dwell time approaches (e.g., [258]) can potentially be used, in the sense that they restrict the time that the system spends in each mode, as we do. However we have no control other the time spent in each mode. Hence, our approach controls the co-simulation step.

6.1.1 Contribution

In this chapter, we develop an approach for preserving the numerical stability of a hybrid co-simulation algorithm.

We make use of a technique that samples the state space at pre-defined points (given by a Poincare surface) and ensures that the state trajectories do not diverge. Inspired by the Joint Spectral Radius theorem [119, 197], and the contribution in Chapter 5, we show how the stability preservation problem can be solved for a restricted class of planar hybrid systems, by reducing it to a problem of finding a non-trivial closed orbit in the transition-delayed system. As an example application of these theoretical results, we provide an algorithm to find such closed orbit in a non-linear hybrid system.

This algorithm can be applied before running co-simulations, to find the appropriate parameterization of transition detection and location schemes, and/or the size of the (co-

)simulation step. Other potential applications include: (i) networked control systems (see, e.g., [113, 381]), where the component responsible for deciding the mode of the plant may be reacting to a delayed signal; and (ii) real-time simulation of hybrid systems, where state transition location can be disabled, or relaxed.

6.1.2 Structure

The next section introduces a motivating example. Then, Section 6.3 formulates the problem that Section 6.4 proposes to solve. The results of this effort are described in Section 6.5. Finally, Section 6.6 discusses related work and Section 6.7 concludes.

6.2 Motivating Example: Relaxed Bouncing Ball Simulation

Consider a bouncing ball modeled with two modes: free-fall, and contact. The free fall mode is dictated by the following ordinary differential equation:

$$\dot{x} = f_2(x) = \begin{bmatrix} x_2 \\ -\frac{d_a}{m} x_2 \left|x_2\right| - g \end{bmatrix}, \tag{6.1}$$

where x_i denotes the i-th component of vector x (e.g., x_2 is the speed), and m, d_a, g denote the mass, air drag, and gravity constants, respectively. The contact mode captures the dynamics of the ball for the brief moments it is in contact with the floor, governed by the following:

$$\dot{x} = f_1(x) = \begin{bmatrix} x_2 \\ \frac{c}{m}(r - x_1) - \frac{d_c}{m} x_2 - \frac{d_a}{m} x_2 \left|x_2\right| - g \end{bmatrix}, \tag{6.2}$$

where d_c, c, r denote the compression damping, stiffness, and the ball radius, respectively.

The ball changes from free fall mode to contact mode when it comes in contact with the floor. Formally, that is when

$$g(x) = \begin{bmatrix} 1 & 0 \end{bmatrix} x - r \leq 0. \tag{6.3}$$

Figure 6.1 shows an example trajectory. The parameters used throughout this document were taken from [128]: $r = 0.25$, $m = 0.650$, $c = 40000$, $d_c = 10$, $d_a = 0.0136$, and $g = 9.81$ (SI Units).

In the same figure, the contact surface (dependent on the radius of the ball) dictates when the mode changes. The moment that the ball changes from the free-fall mode to the contact mode is accurately captured using a state transition location technique ([388]). This enables the correct computation of the state of the ball immediately before entering the contact mode.

One can readily recognize the importance of the state transition location by rendering it inaccurate in the transition from mode *free-fall* to *contact*, thereby introducing a delay in this transition. Figure 6.2 shows an example trajectory where the transition from free-fall

to contact mode is delayed by 0.002s. The execution indicates that the ball reaches the same compression that it had on its initial state. Since the total energy is dissipated via air friction and impact damping, the bouncing ball should tend to the equilibrium. As such, the execution in Figure 6.2 does not constitute a valid execution with respect to the stability of that equilibrium.

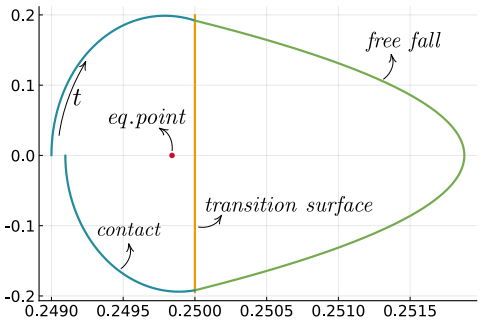


Figure 6.1: Example simulation of the bouncing ball. The horizontal axis refers to position, and vertical refers to velocity. All units are SI.

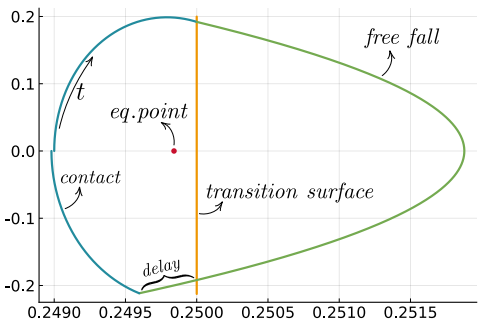


Figure 6.2: Example simulation of the bouncing ball with a transition delay of 0.002s. The horizontal axis refers to position, and vertical refers to velocity. All units are SI.

This experiment hints at the hypothesis that the larger the delay in the transition, the higher the chances that the numerical system does not come to a rest. A natural **research question** then follows: For a given hybrid system whose trajectories eventually tend to an equilibrium, what is the largest delay in the transition that preserves that property? The next section formulates this problem.

6.3 Problem Formulation

We adopt the usual definitions of Hybrid Automata.
Definition 26 (Hybrid automaton [194])**.** A hybrid automaton $\mathcal{S}$ is a collection

$$\mathcal{S} = (Q, E, \mathcal{X}, \mathrm{Dom}, \mathcal{F}, \mathrm{Init}, \mathrm{Guard}, R)$$

where:

- $Q = \{q_1, q_2, \ldots\}$ is a finite set of modes;
- $E \subseteq Q \times Q$ is a finite set of edges called transitions;
- $\mathcal{X} \subseteq \mathbb{R}^n$ is the continuous state space, for some natural n;
- $\mathrm{Dom} : Q \to 2^{\mathcal{X}}$ is the mode domain;
- $\mathcal{F} = \{f_{q_i}(x) : q_i \in Q\}$ is a collection of time-invariant vector fields such that each $f_{q_i}(x)$ is Lipschitz continuous on $Dom(q_i)$;
- $\mathrm{Init} \subseteq Q \times \mathcal{X}$ is a set of initial states;
- $\mathrm{Guard} : E \to 2^{\mathcal{X}}$ defines a guard set for each transition.
- $R : E \times \mathcal{X} \to 2^{\mathcal{X}}$ specifies how the continuous state is reset at each transition by mapping an an edge and old state to a set of possible new states.

Informally, at any time t, $\mathcal{S}$ is in a mode $q_i \in Q$, with a state $x(t)$. The state evolves according to the vector field $\dot{x}(t) = f_{q_i}(x(t))$ associated with mode q_i. $\mathcal{S}$ is allowed to stay in mode q_i for as long as $x(t) \in Dom(q_i)$ holds. $\mathcal{S}$ may switch to mode q_j if $(q_i, q_j) \in E$ and $x(t^-) \in G((q_i, q_j))$. When such switch happens at time t_s, the continuous state is reset to a new continuous state given by $R((q_i, q_j), x(t_s))$. The new state will be the initial state for the vector field associated with new mode.

Remark 6. Note that, for a given unique initial state, the behavior of $\mathcal{S}$ can still be non-deterministic. To see why, note that $\mathcal{S}$ can find itself in a state (mode q_i and continuous state x) that satisfies the mode invariant ($x \in Dom(q_i)$), and the guard of some edge $(q_i, q_j) \in E$ ($x \in G((q_i, q_j))$). In this case, $\mathcal{S}$ may stay in the mode or take the mode transition.

It is common to represent $\mathcal{S}$ as a directed graph with nodes depicting each mode, and edges depicting the transitions. The dynamics associated with each mode are represented inside the corresponding node, and the guard and reset map of each transition are represented near the corresponding edge. The guards are represented with conditions and the resets with assignments of the form $x := \ldots$. When the state is not changed at the transition, that is $x := x$, we omit the assignment. We will often eliminate the time when writing the continuous state x for the sake of simplicity.

Example 25. Figure 6.3 shows the graphical representation of the hybrid automaton for the bouncing ball described in Section 6.2. where $f_1(x)$ is defined in Equation (6.2), $f_2(x)$ is defined in Equation (6.1), and $g(x)$ in Equation (6.3).

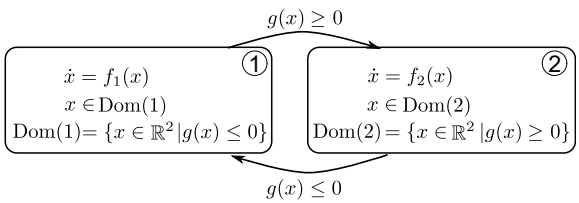


Figure 6.3: Example of bi-modal switched system: bouncing ball.

The Lipschitz continuity assumption in Definition 26 ensures the existence and continuity of a solution inside a mode as shown by Lemma 1.

Definition 27 (Flow)**.** Given a dynamical model $\dot{x}(t) = f(x(t))$, the *flow of the vector field* f, denoted $\phi_f(x, t)$, is the state of the system after a time t if the initial state is x.

Lemma 1 (Flow continuity [341, Theorem 2.1.3, Theorem 2.1.12, Theorem 2.1.13])**.** *Given a dynamical model $\dot{x}(t) = f(x(t))$, if f is Lipschitz-continuous, then $\phi_f(x, t)$ is*

well-defined and Lipschitz-continuous in x and t.

Definition 28 (Hybrid trajectory [148, Definition 2.3])**.** A hybrid trajectory of a system $\mathcal{S}$ is a subset $\mathcal{T} \subseteq Q \times \mathbb{R} \times \mathcal{X}$ with

$$\mathcal{T} = \bigcup_{j=0}^{J-1} \{q_j\} \times \left\{ (t, x) \mid t_j \leq t \leq t_{j+1}, x = \phi_{f_{q_j}}(x_j, t - t_j) \in \mathrm{Dom}(q_j) \right\}$$

for some finite sequence of times $t_0 \leq t_1, \cdots \leq t_J$, modes $q_j \in Q$ and state vector $x_j \in \mathrm{Dom}(q_j)$, such that $(q_{j-1}, q_j) \in E$, $y_j \in \mathrm{Guard}((q_{j-1}, q_j))$ and $x_j \in R((q_{j-1}, q_j), y_j)$ for $j = 1, \ldots, J$, where $y_j = \phi_{f_{q_{j-1}}}(x_{j-1}, t_j - t_{j-1})$.

We define the following notations for trajectories:

$$\begin{aligned}
\mathcal{T}[\underline{t}, \bar{t}] &= \{\, (q, t, x) \in \mathcal{T} \mid \underline{t} \leq t \leq \bar{t} \,\} \\
\mathcal{T}_{\mathcal{X}} &= \{\, x \mid \exists q, t : (q, t, x) \in \mathcal{T} \,\} \\
t_{\min}(\mathcal{T}) &= \inf_{(q,t,x) \in \mathcal{T}} t \\
t_{\max}(\mathcal{T}) &= \sup_{(q,t,x) \in \mathcal{T}} t \\
\mathcal{T}_1 \uplus \mathcal{T}_2 &= \mathcal{T}_1 \cup \{\, (q, t + t_{\max}(\mathcal{T}_1) - t_{\min}(\mathcal{T}_2), x) \mid (q, t, x) \in \mathcal{T}_2 \,\}.
\end{aligned}$$

Note that if $\mathcal{T}$ is a hybrid trajectory then so is $\mathcal{T}[\underline{t}, \bar{t}]$, which allows us to define the combined notation $\mathcal{T}_{\mathcal{X}}[\underline{t}, \bar{t}]$. However, even if $\mathcal{T}_1$ and $\mathcal{T}_2$ are hybrid trajectories, $\mathcal{T}_1 \uplus \mathcal{T}_2$ may not be a hybrid trajectory if for instance it induces a transition that does not respect a guard set.

We say that an equilibrium of a hybrid automaton is *Globally asymptotically stable (GAS)* if all trajectories of the system converge to the equilibrium.

As motivated in Section 6.2, we need to analyze the behavior of a hybrid automaton under delay in the transition. It turns out that we can construct a hybrid automaton for which the stability is equivalent to the stability under delay in the transition of the original system. We present this reduction in the following restricted class of hybrid automata depicted in Figure 6.3.

Definition 29 (Bi-modal Switched Systems)**.** A *Bi-modal Switched* (BMS) system is a hybrid automaton as defined in Definition 26 such that there exists a continuously differentiable function g such that $Q = \{1, 2\}$, $\mathrm{Dom}(1) = \{\, x \mid g(x) \leq 0 \,\}$, $\mathrm{Dom}(2) = \{\, x \mid g(x) \geq 0 \,\}$, $E = \{(1, 2), (2, 1)\}$, $\mathrm{Guard}((1, 2)) = \{\, x \mid g(x) \geq 0 \,\}$, $\mathrm{Guard}((2, 1)) = \{\, x \mid g(x) \leq 0 \,\}$ and $R((1, 2), x) = R((2, 1), x) = \{x\}$ for all $x \in \mathbb{R}^n$.

The hybrid automaton introduced in Example 25, and depicted in Figure 6.3, is a BMS system.

Definition 30. Given a BMS $\mathcal{S}$, we define its transition-delayed $\mathcal{S}_H$ counterpart, with $H > 0$, as in Figure 6.4.

Remark 7. $\mathcal{S}_H$ is non-deterministic, having trajectories where a transition can be delayed by a maximum of H units of time. Moreover, $\mathcal{S}_H$ has the same equilibrium point as the original BMS system. However, the equilibrium point of $\mathcal{S}_H$ may not be GAS, even though the original system $\mathcal{S}$ is GAS, as exemplified in Figure 6.2.

The following results lead to our problem formulation.

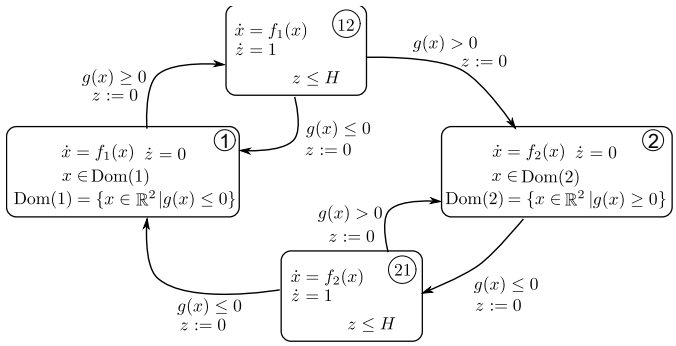


Figure 6.4: Transition delayed switching counterpart of a BMS system (Definition 29). The variable z acts as a clock that is bounded by H.

Proposition 4. *If $\mathcal{T}$ is a trajectory of $\mathcal{S}_H$, then it is also a trajectory of $\mathcal{S}_{H'}$, for any $H' \geq H$.*

Corollary 1. *For any $H' \leq H$, if the equilibrium $x^* \in \mathbb{R}^n$ of $\mathcal{S}_H$ is GAS, then the equilibrium x^* of $\mathcal{S}_{H'}$ is also GAS.*

The previous results naturally prompt the following problem.

Problem 2. *Given a hybrid automaton $\mathcal{S}$ (Definition 26) with a GAS equilibrium x^*, find*

$$\sigma(\mathcal{S}) = \sup_{H \geq 0} H \text{ s.t. } x^* \text{ is a GAS equilibrium of } \mathcal{S}_H \tag{6.4}$$

We call $\sigma(\mathcal{S})$ the Maximum Stable Delay (MSD).

Intuitively, $\sigma(\mathcal{S})$ gives us the maximal delay for which the GAS property x^* is preserved under delayed switching: by Corollary 1, x^* of $\mathcal{S}_H$ is GAS for all $H < \sigma(\mathcal{S})$. Knowing $\sigma(\mathcal{S})$ allows one to define the time step of a simulation algorithm, or the tolerance of a state transition location scheme.

As we have seen, Problem 2 can be reduced to the stability of a hybrid automaton of the form given in Figure 6.4, and thus we can leverage classical tools developed for the stability of hybrid automata. However, deciding the stability of a hybrid automaton is undecidable in general [49], hence this does not provide an *efficient* approach and it only *approximates* the $\sigma(\mathcal{S})$ of Problem 2.

As we show in the following section, under mild assumptions, the stability of the subclass of planar hybrid automata of the form given in Figure 6.4 is equivalent to the existence of a closed orbit. As shown in Section 6.5, this theoretical result can lead to *practical* algorithms to compute $\sigma(\mathcal{S})$ with arbitrary accuracy.

6.4 Orbit and stability

Our approach is similar to the stabilization of a constrained switched system [101, 300]. In the theory of constrained switched systems, the asymptotic stability, can be proven by showing that there are no closed orbits in the system (see [101, Theorem A], [223], and example applications in [152, 154]). We show that there is an equivalence between the

absence of cyclic behaviors in the transition-delayed counterpart (Definition 30), and its asymptotic stability. We start by defining the notion of closed orbit, to formalize cyclic trajectories.

Definition 31 (Closed orbit)**.** Given a trajectory $\mathcal{T}$ as defined in Definition 28, we say that it is a closed orbit if there exists

$$(q_0, t_0, x_0), (\tilde{q}, \tilde{t}, \tilde{x}), (\bar{q}, \bar{t}, \bar{x}) \in \mathcal{T},$$

such that $\tilde{q} \neq \bar{q} = q_0$, $t_0 < \tilde{t} < \bar{t}$ and $\tilde{x} \neq \bar{x} = x_0$.

Remark 8. Given an equilibrium $q^* \in Q, x^* \in \mathrm{Dom}(q)$, the trajectory $\mathcal{T} = \{\, (q^*, t, x^*) \mid 0 \leq t \leq 1 \,\}$ is not a closed orbit as defined in Definition 31, since there does not exist $(\tilde{q}, \tilde{t}, \tilde{x}) \in \mathcal{T}$ such that $\tilde{q} \neq q^*$ and $\tilde{x} \neq x^*$.

Proposition 5. *If there is a closed orbit in $\mathcal{S}_H$, then the equilibrium x^* of $\mathcal{S}_H$ is not GAS.*

The above proposition allows, for a given H, to prove the instability of $\mathcal{S}_H$ by just finding a closed orbit. By trying to find the smallest H for which there is a closed orbit, one can approximate from above the solution $\sigma(\mathcal{S})$ to Problem 2.

However, approximation from above may not be sufficient. To see why, imagine that the smallest H for which there exists a closed orbit in $\mathcal{S}_H$ is found. Then, it might still be the case that there exists a $H' < H$ such that x^* in $\mathcal{S}_{H'}$ is not GAS.

Our contribution is to show that this is impossible, for a class of planar BMS systems, satisfying assumptions 3 to 6, defined in the following.

For that we use classical results of analysis of planar non-linear systems such as continuity and monotonicity of a *Poincaré Map*. Such classical notions need to be carefully used in this setting since they are usually developed for non-hybrid systems.

We start by discussing the monotonicity. We will see in Corollary 2 that the Poincaré Map is monotonous for the hybrid automaton $\mathcal{S}$. However, for a non-deterministic hybrid automaton such as $\mathcal{S}_H$, monotonicity of the Poincaré Map is not guaranteed as two trajectories can intersect as long as they are in different modes. Nevertheless, the topological argument commonly used to prove monotonicity (see e.g. [183, Section 10.4]) can still be used to obtain the result given in Lemma 2 for planar BMS systems.

We start by defining the following connectedness concepts from topology; see e.g. [265, Chapter 3].

Definition 32 (Path)**.** Given a set X and two points $x, y \in X$, a *path* in X from x to y is the image of a continuous map $f : [0, 1] \to X$ such that $f(0) = x$ and $f(1) = y$.

We denote the union of all paths in X from x to y as $[x, y]_X$.

Definition 33 (Path components of X)**.** We define an equivalence relation on the set X by defining $x \sim y$ if there is a path in X from x to y. The equivalence classes are called the *path components* of X.

Given a set U, we denote its *closure* by $\overline{U}$ and its relative interior by $\mathrm{relint}(U)$.

We denote the switching surface as

$$G = \{\, x \mid g(x) = 0 \,\}.$$

To avoid pathological cases, we assume that the equilibrium $x^* \notin G$. Without loss of

generality, we consider that the equilibrium is in the interior of $\mathrm{Dom}(1)$ and assume that it is GAS for the mode 1.

Given the vector field f_1 of the first mode of a planar BMS $\mathcal{S}$ defined in Definition 29, we assume the existence of a continuously differentiable function $s : \mathbb{R}^2 \to \mathbb{R}$ such that the curve

$$S_p = \{\, x \mid s(x) = 0, \nabla s(x) \cdot f_1(x) > 0 \,\} \tag{6.5}$$

satisfies $S_p \cap G = \emptyset$. We refer to the closed section S_p as Poincaré curve, its definition is similar to the notion of *local sections*; see [183, Section 10.2].

The following assumption constraints the Poincaré curve to be "connected" and 'moving towards the origin".

Assumption 3. *The Poincaré curve S_p has a single path component, $\overline{S_p} = S_p \cup \{x^*\}$ and the restriction of the euclidean norm to S_p is injective.*

We use the following notations for the *return time* and *Poincaré Map*, given a vector field f and a set U:

$$\tau_U^f(x) = \inf\{\, t \mid t > 0, \phi_f(x,t) \in U \,\},$$
$$P_U^f(x) = \phi_f(x, \tau_U^f(x)).$$

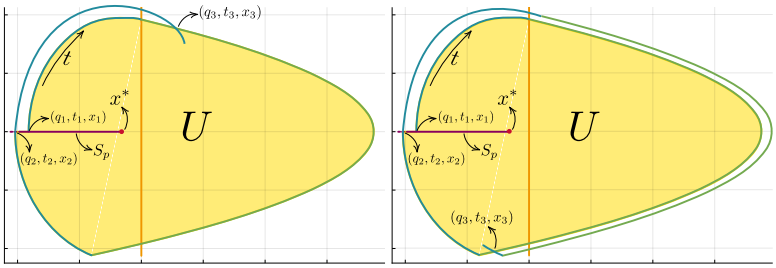
Note that $P_U^f(x)$ is not defined when $\tau_U^f(x)$ is infinite.

Given a trajectory $\mathcal{T}$, a set U and a time $t_0 \in \mathbb{R}$, we define the following notation

$$\tau_U^{\mathcal{T}}(t_0) = \inf\{\, t \mid (q,t,x) \in \mathcal{T}, t > t_0, x \in U \,\},$$
$$\mathcal{P}_U^{\mathcal{T}}(t_0) = \{\, (q,t,x) \in \mathcal{T} \mid t = \tau_U^{\mathcal{T}}(t_0) \,\}.$$

Remark 9. Given a hybrid automaton $\mathcal{S}$ and a trajectory $\mathcal{T}$, if $\mathcal{P}_G^{\mathcal{T}}(0)$ is defined, then it is a set of points representing the different modes the system undergoes at the switching surface (recall Definition 28).

The following lemma shows that, whenever a trajectory $\mathcal{T}$ of $\mathcal{S}_H$ intersects with itself, it must be under a different mode. Figures 6.5a and 6.5b give representative examples of this situation.



(a) (b)

Figure 6.5: Illustrations of Lemma 2. In both cases, the intersection, caused by the delayed switching, happens under different modes, and never in the same mode. Each axis refer to the dimensions in the state-space.

Lemma 2. *Given a planar hybrid automaton $\mathcal{S}_H$ as defined in Definition 30, with a Poincaré curve S_p as defined in Equation* (6.5) *and a trajectory $\mathcal{T}$ of $\mathcal{S}_H$, consider $(q_1, t_1, x_1) \in \mathcal{T}$ with $x_1 \in S_p$ and $(q_2, t_2, x_2) \in \mathcal{P}^{\mathcal{T}}_{S_p}(t_1)$ with $\|x_2 - x^*\|_2 \geq \|x_1 - x^*\|_2$. Under Assumption 3, there exists a path component U of $\mathcal{X} \setminus (\mathcal{T}_{\mathcal{X}}[t_1, t_2] \cup [x_2, x_1]_{S_p})$ such that:*

(1) $x \in \overline{U}$ for all $x \in S_p$ such that $\|x - x^*\|_2 \leq \|x_2 - x^*\|_2$; and
(2) there exists $\epsilon > 0$ such that $\mathcal{T}_{\mathcal{X}}[t_2, t_2 + \epsilon] \cap U = \emptyset$.

Moreover, if $\mathcal{T}_{\mathcal{X}}[t_2, \infty] \cap U \neq \emptyset$ then the intersection $(q_3, t_3, x_3) \in \mathcal{P}^{\mathcal{T}}_{\overline{U}}(t_2)$ is such that there exists $(q_4, t_4, x_3) \in \mathcal{T}[t_1, t_2]$ with $q_3 \neq q_4$.

Proof. Let U be the path component of $\mathcal{X} \setminus (\mathcal{T}_{\mathcal{X}}[t_1, t_2] \cup [x_2, x_1]_{S_p})$, that contains $x^* \in U$. By Assumption 3, there is a path in $\overline{S_p}$ from x^* to x_2. This path is contained in $\overline{U}$ as it cannot intersect the interior of $\mathcal{T}_{\mathcal{X}}[t_1, t_2]$ by definition of t_2. Therefore, by the injectiviy assumption of Assumption 3, the set U must contain all $x \in S_p$ such that $\|x - x^*\|_2 \leq \|x_2 - x^*\|_2$.

As $\nabla s(x) \cdot f_1(x) > 0$ for all $x \in S_p$ (see Equation (6.5)) and $\|x_2 - x^*\|_2 \geq \|x_1 - x^*\|_2$, there exists $\epsilon > 0$ such that $\mathcal{T}_{\mathcal{X}}[t_2, t_2 + \epsilon] \cap U = \emptyset$. As the trajectory at $t_2 + \epsilon$ is not in U, if $\mathcal{T}_{\mathcal{X}}[t_2, \infty] \cap U \neq \emptyset$ then $(q_3, t_3, x_3) \in \mathcal{P}^{\mathcal{T}}_{\overline{U}}(t_2)$ is defined. The point x_3 cannot belong to $[y, x]_{S_p}$ as $\nabla s(x) \cdot f_1(x) > 0$ hence $x_3 \in \mathcal{T}_{\mathcal{X}}[t_1, t_2]$. For any $(q, t, x_3) \in \mathcal{T}[t_1, t_2]$, we must have $q \neq q_3$ since by Lemma 1, two trajectories in the same mode cannot intersect. □

As a corollary, the monotonicity of the Poincaré Map holds for the mode 1, as stated in Corollary 2. Additionally, the monotonicity also holds for the original hybrid automaton without delay $\mathcal{S}$, because, for two trajectories to intersect, they must be in the same domain hence the same mode (except at the switching surface, they could be in different mode but that is during a time interval of measure 0). However, when applying the same result to the delayed hybrid automaton, care must be taken to consider only trajectories that remain in mode 1, as these are not affected by the delayed switching. If a trajectory of the delayed hybrid automaton remains in the domain of mode 1 between two consecutive intersections with the Poincaré Map, Corollary 2 shows that the second intersection has a smaller euclidean norm. In fact, as shown by Lemma 3, this implies that the trajectory will remain in mode 1 until its end.

Corollary 2. *Consider a planar hybrid automaton $\mathcal{S}_H$ as defined in Definition 30 and a Poincaré curve S_p as defined in Equation* (6.5)*. For all $x \in S_p$, if for any t, $\phi_f(x, t) \in \mathrm{Dom}(1)$, then $\left\| P^{f_1}_{S_p}(x) - x^* \right\|_2 < \|x - x^*\|_2$.*

Lemma 3. *Given a delay H and a planar hybrid automaton $\mathcal{S}_H$ as defined in Definition 30, with a Poincaré curve S_p as defined in Equation* (6.5)*, and a trajectory $\mathcal{T}$ of $\mathcal{S}_H$, consider $(1, t_1, x_1) \in \mathcal{T}$ with $x_1 \in S_p$ and $(1, t_2, x_2) \in \mathcal{P}^{\mathcal{T}}_{S_p}(t_1)$. Under Assumption 3, if $\mathcal{T}_{\mathcal{X}}[t_1, t_2] \subseteq \mathrm{Dom}(1)$, then $\mathcal{T}_{\mathcal{X}}[t_2, \infty] \subseteq \mathrm{Dom}(1)$.*

Proof. Let U be the path component of $\mathcal{X} \setminus (\mathcal{T}_{\mathcal{X}}[t_1, t_2] \cup [x_2, x_1]_{S_p})$ containing the equilibrium x^*. If $\mathcal{T}_{\mathcal{X}}[t_2, \infty] \not\subseteq \overline{U}$ then given

$$(q', t', x') \in \mathcal{P}^{\mathcal{T}}_{\mathcal{T}_{\mathcal{X}}[t_1, t_2] \cup [x_2, x_1]_{S_p}}(t_2),$$

we know that $x' \in \mathcal{T}_{\mathcal{X}}[t_1, t_2]$ since $\nabla s(x) \cdot f_1(x) > 0$ for all $x \in S_p$. As $\mathcal{T}_{\mathcal{X}}[t_1, t_2] \subseteq \mathrm{Dom}(1)$, $q' = 1$ which is impossible by Lemma 1. Therefore $\mathcal{T}_{\mathcal{X}}[t_2, \infty] \subseteq \overline{U} \subseteq \mathrm{Dom}(1)$.

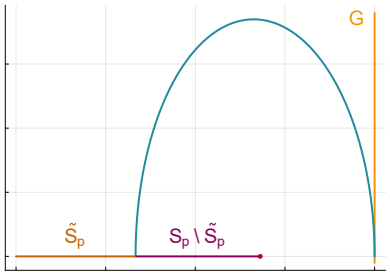


Figure 6.6: Illustration of the set $\tilde{S}_p$, defined in Equation (6.6), for the bouncing ball example. The horizontal axis refers to position, and vertical refers to velocity. In blue is represented the trajectory starting at the point $\bar{x} \in S_p$ such that the trajectory is tangent to the switching surface G represented in yellow. All trajectories starting at a point $x \in S_p$ such that $\|x - x^*\|_2 > \|\bar{x} - x^*\|_2$ eventually cross the switching surface, these points constitute the set $\tilde{S}_p$, represented in brown.

□

In view of Corollary 2 and Lemma 3, if a trajectory does not intersect the switching surface between two intersections of the Poincaré curve (hence staying in mode 1) then it will repeat this behavior indefinitely. Therefore, as our aim is to find a closed orbit, we restrict our attention to points of the Poincaré curve from which a trajectory of $\mathcal{S}$ intersects the switching surface. See Figure 6.6. We denote the set of such points as follows:

$$\tilde{S}_p = \{\, x \in S_p \mid \tau_G^{f_1}(x) \text{ is finite}, \tau_G^{f_1}(P_G^{f_1}(x)) > 0 \,\}. \tag{6.6}$$

Since the equilibrium x^* is GAS for the mode 1, by Corollary 2 and Lemma 1, any trajectory of $\mathcal{S}_H$ (independently of H) starting at a point in $S_p \setminus \tilde{S}_p$ will converge to x^*. Lemma 4 provides a converse result, that is, if x^* is not GAS and $\mathcal{S}_H$ does not admit any closed orbit, then subsequent intersections with S_p will be farther from the origin.

However, extra care must be taken of where to place the Poincaré curve, to make sure all its intersections with S_p happen in mode 1. Moreover, we need to assume that the dynamics of mode 2, under the domain of mode 1, do not prevent these intersections (e.g., it could lead to chattering [236, Section 1.2.4] around the switching surface, because any trajectory starting in mode 2 will cross the switching surface to mode 1, by the stability of x^* in $\mathcal{S}$).

The following assumption handles these scenarios. Figure 6.7 illustrate the concepts in this assumption. Moreover, they show the impact of different Poincaré curves.

Assumption 4. *Given a planar hybrid automaton $\mathcal{S}_H$, let*

$$\begin{aligned} \tilde{G} &= \{\, x \in G \mid \nabla g(x) \cdot f_2(x) < 0 \,\}, \\ T_2 &= \inf_{x \in \tilde{G}} \tau_{S_p}^{f_2}(x), \text{ and} \\ D &= \{\, \phi_{f_2}(x, h) \mid x \in \tilde{G}, 0 \le h < T_2 \,\}. \end{aligned} \tag{6.7}$$

We assume that for every point $x \in D$, $\tau_{S_p}^{f_1}(x)$ is finite and smaller than $\tau_G^{f_1}(x)$ (which may be infinite if the system never crosses the switching surface).

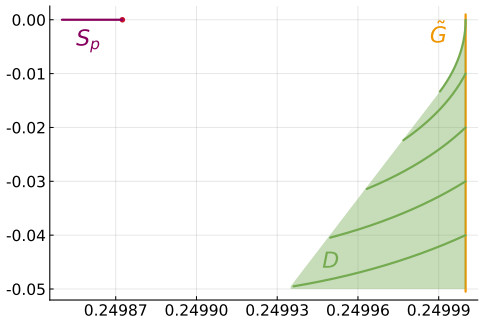


Figure 6.7: Illustration of Assumption 4 for the bouncing ball example introduced in Section 6.2. The horizontal axis refers to position, and vertical refers to velocity. Note that, in this example, by choosing $s(x) = x_2$, the value T_2 is infinite. Therefore, the requirement $\sigma(\mathcal{S}) < T_2$ of Theorem 4 is automatically satisfied. In the figure, we represent the set D for a value of $T_2 = 0.0005$.

Under assumption 4, and a sufficiently small H in $\mathcal{S}_H$, trajectories that start in $\tilde{S}_p$ acquire a predictable pattern, illustrated in Figure 6.8. We now define the notation and restrictions to represent these trajectories.

Given a point $x_0 \in \tilde{S}_p$, $0 \leq h_1, h_2$, let

$$x_1 = P_G^{f_1}(x) \qquad x_2 = \phi_{f_1}(x_1, h_1) \tag{6.8}$$

$$x_3 = P_G^{f_2}(x_2) \qquad x_4 = \phi_{f_2}(x_3, h_2). \tag{6.9}$$

If $h_1 < \tau_G^{f_1}(x_1)$ and $h_2 < T_2$, by Assumption 4, the following trajectory is admitted by planar $\mathcal{S}_{\max(h_1,h_2)}$.

$$\mathcal{TQ}(x, h_1, h_2) = \mathcal{T}P_G^{f_1}(x) \uplus \mathcal{T}\phi_{f_1}(h_1, x_1) \uplus \mathcal{T}P_G^{f_2}(x_2) \tag{6.10}$$

$$\uplus\, \mathcal{T}\phi_{f_2}(h_2, x_3) \uplus \mathcal{T}P_{S_p}^{f_1}(x_4). \tag{6.11}$$

We also introduce the notation

$$\mathcal{Q}(x, h_1, h_2) \triangleq P_{S_p}^{f_1}(x_4),$$

illustrated in Figure 6.8.

The following lemma is illustrated in Figure 6.9.

Lemma 4. *Given a delay $H < T_2$ (where T_2 is defined in Equation* (6.7)*), a planar hybrid automaton $\mathcal{S}_H$ as defined in Definition 30, with a Poincaré curve S_p as defined in Equation* (6.5)*. Under assumptions 3 and 4, if the equilibrium x^* is not GAS for $\mathcal{S}_H$, and $\mathcal{S}_H$ does not admit any closed orbit, then there exists two points $x, y \in \tilde{S}_p$ and delays $0 \leq h_1, h_2 \leq H$ such that $y = \mathcal{Q}(x, h_1, h_2)$ and $\|y - x^*\|_2 > \|x - x^*\|_2$.*

Proof. Since $\mathcal{S}_H$ is not GAS, it admits a trajectory $\mathcal{T}$ that does not converge to the equilibrium x^*.

The proof is divided into two parts: first we prove that trajectory $\mathcal{T}$ contains infinitely many mode transitions. Then we use that fact to prove that successive intersections satisfy

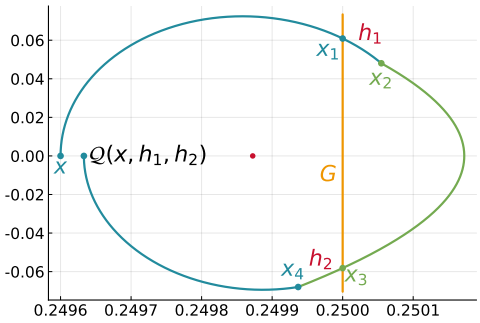


Figure 6.8: Illustration of the mapping $\mathcal{Q}(x, h_1, h_2)$ for the bouncing ball example introduced in Section 6.2. The x-axis and y-axis correspond respectively to the position and velocity. The values used for this figure are $x = 0.2496$ and $h_1 = h_2 = 0.001$.

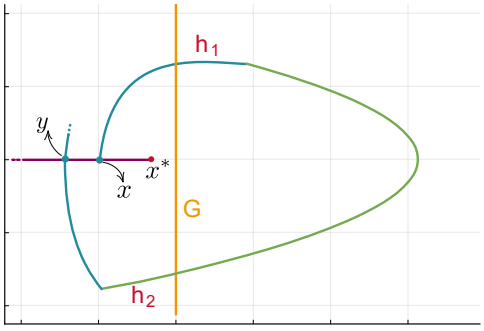


Figure 6.9: Illustration of Lemma 4 for the bouncing ball example. The horizontal axis refers to position, and vertical refers to velocity. Since the equilibrium x^* is not GAS for $\mathcal{S}_H$, and $\mathcal{S}_H$ does not admit any closed orbit, the figure shows a trajectory that Lemma 4 proves to exist.

the claim in the lemma. In the second part, we make use of a result that is proved later (Lemma 7). Lemma 7 does not depend on Lemma 4.

We now prove by contradiction that trajectory $\mathcal{T}$ contains infinitely many mode transitions. If it contains only finitely many transitions, there is a mode q_∞ and time t_∞ such that, for all $(q, t, x) \in \mathcal{T}$ with $t > t_\infty$, we have $q = q_\infty$. We cannot have $q_\infty = 1$ as the equilibrium x^* is GAS for mode 1. Similarly, we cannot have $q_\infty = 2$ since the equilibrium x^* is GAS for $\mathcal{S}$. In this case, $(2, t', x') \in \mathcal{P}_G^{\mathcal{T}}(t_\infty)$ is defined. Let $(2, t'', x'') \in \mathcal{P}_G^{\mathcal{T}}(t')$. Then, by Assumption 4, there must be t''', x''' such that $(1, t''', x''') \in \mathcal{T}[t', t'']$. This concludes the proof that trajectory $\mathcal{T}$ contains infinitely many mode transitions.

Now we focus on the second part of the proof. By Assumption 4, after each transition from mode 2 to mode 1, the trajectory $\mathcal{T}$ must intersect the Poincaré curve before it transitions from mode 1 to mode 2. Therefore the trajectory intersects the Poincaré curve an infinite amount of times and is in mode 1 each time it intersects it. Let $(1, t_1, x_1), (1, t_2, x_2), \ldots \in \mathcal{T}$ with $t_1 < t_2 < \cdots$ and $x_1, x_2, \ldots \in S_p$ be this sequence intersections.

There are no $i \neq j$ such that $\|x_i - x^*\|_2 = \|x_j - x^*\|_2$ because, by Assumption 3,

$\|x_i - x^*\|_2 = \|x_j - x^*\|_2$ implies that $x_i = x_j$ which contradicts the statement that no closed orbit is admitted. There are two possible cases.

- The sequence $(\|x_i - x^*\|_2)_i$ is decreasing with i. Since it is bounded below by 0, it converges. Let $n = \lim_{i\to\infty} \|x_i - x^*\|_2$. Since the trajectory is not stable, $n > 0$. By Assumption 3, there exists a unique $x \in S_p$ such that $\|x - x^*\|_2 = n$. By Lemma 7, $\mathcal{S}_H$ admits a closed orbit containing x which contradicts the statement. Hence this case cannot be true.
- There exists a k such that $\|x_{k+1} - x^*\|_2 > \|x_k - x^*\|_2$. Let

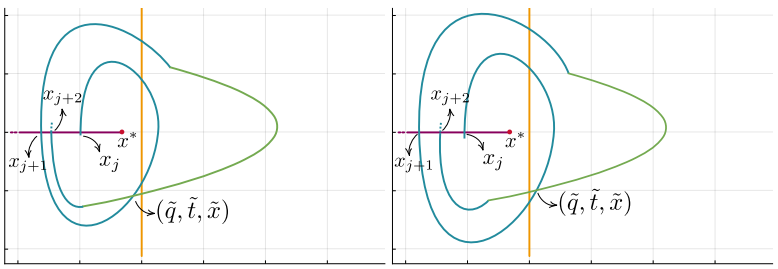
$$K = \{\, k \mid \|x_{k+1} - x^*\|_2 > \|x_k - x^*\|_2 \,\}.$$

  There are two possible sub-cases.
  - If there exists $k \in K, t \in \mathbb{R}, x \in \mathcal{X}$ such that $(2, t, x) \in \mathcal{T}[t_k, t_{k+1}]$ then we are done.
  - Otherwise, for all $k \in K$, $x_{k+1} = P^{f_1}_{S_p}(x_k)$. Note that, by Corollary 2, it must be the case that the trajectory between x_k and x_{k+1} is going into the domain of mode 2.
    Let $j \in K$ be such that $j + 1 \notin K$, and

$$(\tilde{q}, \tilde{t}, \tilde{x}) \in \mathcal{P}^{\mathcal{T}}_{\mathcal{T}_{\mathcal{X}}[t_j, t_{j+1}]}(t_{j+1}),$$

    as illustrated in Figures 6.10a and 6.10b. By Lemma 2, $\tilde{q} = 2$. Let $\bar{t}$ be such that $(1, \bar{t}, \tilde{x}) \in \mathcal{T}[t_j, t_{j+1}]$. If $\tilde{x} \in \mathrm{Dom}(1)$, as in Figure 6.10a, then $\mathcal{T}[\bar{t}, \tilde{t}]$ is a closed orbit which contradicts the statement. Otherwise, $\tilde{x} \in \mathrm{Dom}(2)$[2], as shown in Figure 6.10b. Let U be the path component of $\mathcal{X} \setminus \mathcal{T}_{\mathcal{X}}[\bar{t}, \tilde{t}]$ containing the points $x \in S_p$ such that $\|x - x^*\|_2 < \|x_{j+1} - x^*\|_2$ and $j' = \min\{\, k \in K \mid k > j + 1, \|x_{k+1} - x^*\|_2 > \|x_{j+1} - x^*\|_2 \,\}$. Since $x_{j'+1} \notin \overline{U}$, by Lemma 1, $(q', t', x') \in \mathcal{P}^{\mathcal{T}}_{\mathcal{T}_{\mathcal{X}}[\bar{t}, \tilde{t}]}(t_{j'})$ is defined and is such that $t_{j'} \le \bar{t} \le t_{j'+1}$, $q' = 1$ and there exists $(2, t'', x') \in \mathcal{T}[\bar{t}, \tilde{t}]$. The trajectory $\mathcal{T}[t'', t']$ is a closed orbit which contradicts the statement.

□



(a) (b)

Figure 6.10: Illustrations of the second sub-case of Lemma 4. The axis refer to the dimensions in the state-space.

[2] In this case, $\mathcal{T}[\bar{t}, \tilde{t}]$ is not a closed orbit since it requires a transition between mode 2 and mode 1 at $\tilde{t}$ which is not possible since $\bar{x} \in \mathrm{Dom}(2)$.

The following lemma shows that by adjusting the delay in $\mathcal{S}_H$, one can find admissible trajectories whose successive intersections with the Poincaré curve are getting closer to the equilibrium. The lemma statement is illustrated in Figure 6.11.

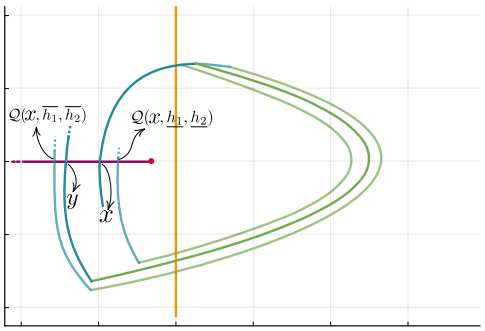


Figure 6.11: Illustrations of Lemma 5. The axis refer to the dimensions in the state-space.

Lemma 5. *Given a delay $H < T_2$ (where T_2 is defined in Equation (6.7)), a planar hybrid automaton $\mathcal{S}_H$ as defined in Definition 30, with a Poincaré curve S_p as defined in Equation (6.5). Under Assumption 3, if there exist two points $x, y \in \tilde{S}_p$ and delays $0 \leq h_1, h_2 \leq H$ such that $y = \mathcal{Q}(x, h_1, h_2)$ and $\|y - x^*\|_2 > \|x - x^*\|_2$ then there exist delays $0 \leq \overline{h_1}, \underline{h_1} \leq h_1$ and $0 \leq \overline{h_2}, \underline{h_2} \leq h_2$ such that*

$$x \in [\mathcal{Q}(x, \underline{h_1}, \underline{h_2}), \mathcal{Q}(x, \overline{h_1}, \overline{h_2})]_{S_p}.$$

Proof. Consider a trajectory $\mathcal{T}$ of $\mathcal{S}$ starting at y. Let $y_0 = y$, $t_0 = 0$ and

$$(1, t_{k+1}, y_{k+1}) \in \mathcal{P}^{\mathcal{T}}_{S_p}(t_k), \tag{6.12}$$

for $k = 0, 1, \ldots$ This sequence is illustrated in Figure 6.12. By Corollary 2, the sequence $(\|y_k - x^*\|_2)_k$ is non-increasing in k. Let $j = \inf\{\, k \mid \|y_k - x^*\|_2 < \|x - x^*\|_2 \,\}$, we know that $j < \infty$ since the equilibrium x^* is GAS for $\mathcal{S}$. In Figure 6.12, $j = 3$. We show by recurrence the existence of a sequence of delays $h_{1,k} \leq h_1$ and $h_{2,k} \leq h_2$ for $k = 0, 1, \ldots, j$ such that $y_k = \mathcal{Q}(x, h_{1,k}, h_{2,k})$, where y_k is defined in Equation (6.12). We set $h_{1,0} = h_1$ and $h_{2,0} = h_2$. For $k \geq 0$, given $h_{1,k} \leq h_1$ and $h_{1,k} \leq h_2$, let

$$(q', t', x') \in \mathcal{P}^{\mathcal{T}}_{\mathcal{T}_{\mathcal{X}}\mathcal{Q}(x,h_{1,k},h_{2,k})}(t_k),$$

as illustrated in Figure 6.12, and

$$(q'', t'', x') \in \mathcal{T}\mathcal{Q}(x, h_{1,k}, h_{2,k}),$$

where $\mathcal{T}\mathcal{Q}(x, h_{1,k}, h_{2,k})$ is defined in Equation (6.10). By Lemma 2, we only have the following two cases to consider[3]: if $q' = 2$, $q'' = 1$ and $x' \in \mathrm{Dom}(2)$ then we set $h_{2,k+1} = 0$ and $h_{1,k+1} = t'' - \tau^{f_1}_G(x)$; otherwise, if $q' = 1$, $q'' = 2$ and $x' \in \mathrm{Dom}(1)$ then we set $h_{1,k+1} = h_{1,k}$ and $h_{2,k+1} = t'' - (\tau^{f_1}_G(\underline{x}) + h_{1,k} + \tau^{f_2}_G(\phi_{f_1}(P^{f_1}_G(x), h_{1,k})))$. The lemma is proved with $\overline{h_1} = h_{1,j-1}$, $\underline{h_1} = h_{1,j}$, $\overline{h_2} = h_{2,j-1}$ and $\underline{h_2} = h_{2,j}$. □

[3]Note that in both cases, we have $\mathcal{T}\mathcal{Q}(x, h_{1,k+1}, h_{2,k+1}) = \mathcal{T}\mathcal{Q}(x, h_{1,k}, h_{2,k})[0, t''] \uplus \mathcal{T}[t', t_{k+1}]$.

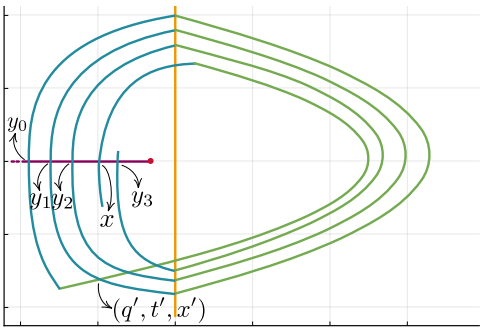


Figure 6.12: Illustration of the sequence defined in Equation (6.12).

From the existence of an unstable trajectory, Lemma 4 combined with Lemma 5 ensures the existence of a point x such that for different delays, the next intersection with the Poincaré curve is either "above" or "below" it. We now prove the continuity of the Poincaré Map to show that there are delays such that the next intersection is exactly x.

The continuity of Poincaré Map is classically deduced from the *Implicit Function Theorem*, see e.g. [313, Section 5.2, Theorem 2.1]:

Theorem 3 (Implicit Function Theorem)**.** *Consider a continuously differentiable function $F : \mathbb{R}^{n+1} \to \mathbb{R}$ and $x_0 \in \mathbb{R}^n, y_0 \in \mathbb{R}$ such that $\frac{\partial F}{\partial y}(x_0, y_0)$ is non-zero. Then there exists an open set $U \subseteq \mathbb{R}^n$ containing x_0 such that there exists a unique continuously differentiable function $h : U \to \mathbb{R}$ such that $h(x_0) = y_0$ and $F(x, h(x)) = F(x_0, y_0)$ for all $x \in U$.*

Lemma 6 (Continuity of Poincaré Map)**.** *Consider two continuously differentiable functions $f, g : \mathbb{R}^2 \to \mathbb{R}$, and let $G = \{ x \mid g(x) = 0 \}$. Consider any open set A for which $A \cap G = \emptyset$. If for all $x \in A$, $z = P_G^f(x)$ is defined, and $\nabla g(z) \cdot f(z) \neq 0$, then the Poincaré Map $P_G^f(x)$ restricted to A is continuous.*

Proof. Consider a point $x_0 \in A$. Let $t_0 = \tau_G^f(x_0)$ and $F(x, t) = g(\phi_f(x, t))$, we have $F(x_0, t_0) = 0$. Furthermore

$$\frac{\partial F}{\partial t}(x_0, t_0) = \nabla g(\phi_f(x_0, t_0)) \cdot \frac{\partial \phi_f(x_0, t_0)}{\partial t} = \nabla g(\phi_f(x_0, t)) \cdot f(\phi_f(x_0, t_0))$$

which is nonzero by assumption. Therefore, by Theorem 3, there exists an open set $U \subseteq \mathbb{R}^2$ containing x_0 and a unique continuously differentiable function h such that $F(x, h(x)) = 0$ for each $x \in U$. By the unicity of h, we know that $P_G^f(x) = \phi_f(x, h(x))$ for each $x \in U$. Since the function $h(x)$ is continuous in x and the function ϕ is continuous in x and t by Lemma 1, the Poincaré Map $P_G^f(x)$ is continuous in x_0. □

Remark 10. The continuity of the Poincaré Map cannot be readily generalized to the hybrid automaton context. Given planar $\mathcal{S}_H$, the Poincaré Map may have discontinuities in x and in the delay in the transition. Discontinuities in x may happen if the trajectory is tangent to the switching surface while discontinuities in the delay in the transition may happen when the time spent in $\mathrm{Dom}(2)$ is exactly the delay in the transition. Figures 6.13a and 6.13b illustrate these examples.

While the Poincaré Map is not continuous everywhere, under some assumptions that will be stated, our argument only relies on its continuity on regions where it is locally continuous.

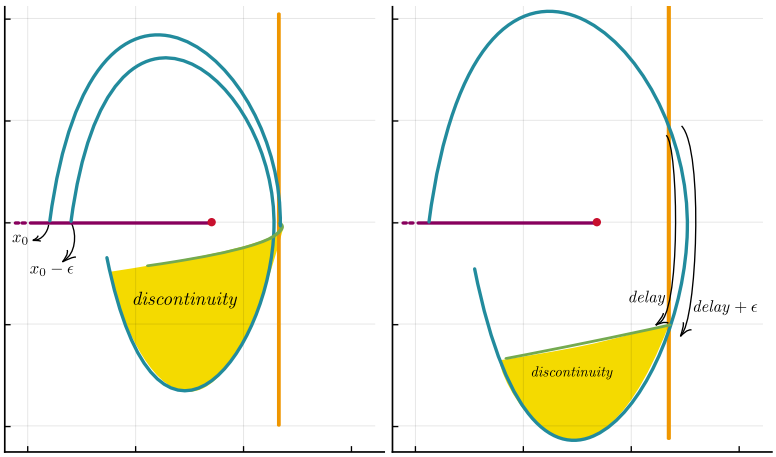


Figure 6.13: Illustrations of Remark 10. Discontinuities in x may happen if the trajectory is tangent to the switching surface (a), and discontinuities in the delay in the transition may happen when the time spent in $\mathrm{Dom}(2)$ is exactly the delay in the transition (b). The axis refer to the dimensions in the state-space.

In view of the statement of Lemma 6, we add the following two assumptions.

Assumption 5. *We assume that for every point $x \in \mathrm{relint}(\mathrm{Dom}(2))$, the point $y = P_G^{f_2}(x)$ is such that the surface normal $\nabla g(y)$ and $f_2(y)$ are not perpendicular, that is, $\nabla g(y) \cdot f_2(y) \neq 0$.*

Assumption 6. *We assume that for every point $x \in \tilde{S}_p$, the point $y = P_G^{f_1}(x)$ is such that the surface normal $\nabla g(y)$ and $f_1(y)$ are not perpendicular, that is, $\nabla g(y) \cdot f_1(y) \neq 0$.*

Lemma 7. *Consider a planar BMS $\mathcal{S}$ as defined in Definition 29 satisfying Assumption 5, and a Poincaré curve S_p satisfying Assumptions 4 and 6. For any point $x_0 \in \tilde{S}_p$, the Poincaré Map $\mathcal{Q}(x_0, h_1, h_2)$ is defined and continuous with respect to x_0, $h_1 \in [0; \tau_G^{f_1}(P_G^{f_1}(x))[$ and $h_2 \in [0; T_2[$ where T_2 is defined in Equation* (6.7).

Proof. Let x_1, x_2, x_3, x_4 be as defined in Equations (6.8) and (6.9) and x_5 denote $\mathcal{Q}(x, h_1, h_2)$.

By Assumption 4 and Lemma 6, x_5 depends continuously on x_4 and by Lemma 1, x_4 depends continuously on h_2 hence x_5 depends continuously on h_2.

By Lemma 1, x_4 depends continuously on x_3 and by Assumption 5 and Lemma 6, x_3 depends continuously on x_2 hence x_5 depends continuously on x_2.

By Lemma 1, x_2 depends continuously on h_1 hence x_5 depends continuously on h_1.

By Lemma 1, x_2 depends continuously on x_1 and by Assumption 6 and Lemma 6, x_1 depends continuously on x_0 hence x_5 depends continuously on x_0.

□

Remark 11. Continuity of trajectories with respect to the state x is similar to *incremental stability*, see e.g. [146] for a definition of this concept. However, incremental stability is a stronger notion as it requires contraction in addition to continuity.

Proposition 6. *Given a delay $H < T_2$ (where T_2 is defined in Equation* (6.7)*), a planar hybrid automaton $\mathcal{S}_H$ as defined in Definition 30, with a Poincaré curve S_p as defined in Equation* (6.5)*. Under assumptions 3 to 6, If the equilibrium $x^* \in \mathbb{R}^n$ of $\mathcal{S}_H$ is not GAS and $H < T_2$, then $\mathcal{S}_H$ admits a closed orbit.*

Proof. Suppose by contradiction that there exists H such that $\mathcal{S}_H$ is not GAS but does not admit any closed orbit. Consider one such value of H. By Lemma 4 and Lemma 5, there a point $x \in \mathbb{R}^n$, delays $0 \leq \overline{h_1}, \underline{h_1} \leq H$ and $0 \leq \overline{h_2}, \underline{h_2} \leq H$ such that

$$x \in [\mathcal{Q}(x, \underline{h_1}, \underline{h_2}), \mathcal{Q}(x, \overline{h_1}, \overline{h_2})]_{S_p}.$$

By Lemma 7, there exists $\min(\overline{h_1}, \underline{h_1}) \leq h_1 \leq \max(\overline{h_1}, \underline{h_1})$ and $\min(\overline{h_2}, \underline{h_2}) \leq h_2 \leq \max(\overline{h_2}, \underline{h_2})$ such that $\mathcal{Q}(x, h_1, h_2) = x$. This contradicts the fact that $\mathcal{S}_H$ admits no closed orbit. Hence the proof is complete. □

The above claim allows us to find the maximum switching delay $\sigma(\mathcal{S})$ (solution to Problem 2) by solving the following problem:

$$\hat{\sigma}(\mathcal{S}) = \inf_{H \geq 0} H \text{ s.t. } \mathcal{S}_H \text{ admits a closed orbit.} \tag{6.13}$$

Theorem 4. *Given a planar BMS $\mathcal{S}$ as defined in Definition 29 satisfying Assumption 5 and a Poincaré curve S_p as defined in Equation* (6.5) *satisfying Assumptions 3, 4 and 6, and $\sigma(\mathcal{S}) < T_2$ (where T_2 is defined in Equation* (6.7)*), the identity $\sigma(\mathcal{S}) = \hat{\sigma}(\mathcal{S})$ holds.*

Proof. The inequality $\sigma(\mathcal{S}) \leq \hat{\sigma}(\mathcal{S})$ follows from Proposition 5 and we show $\sigma(\mathcal{S}) \geq \hat{\sigma}(\mathcal{S})$ by contradiction. If $\sigma(\mathcal{S}) < \hat{\sigma}(\mathcal{S})$ then there exists a delay H such that $\sigma(\mathcal{S}) < H < \hat{\sigma}(\mathcal{S})$. Since $\sigma(\mathcal{S}) < H$, the equilibrium x^* is not GAS for $\mathcal{S}_H$. Therefore, by Proposition 6, $\mathcal{S}_H$ admits a closed orbit. This is in contradiction with $H < \hat{\sigma}(\mathcal{S})$. □

6.5 Results

This section shows how Theorem 4 can be applied to compute the MSD $\sigma(\mathcal{S}^{wc})$ in Problem 2, for the bouncing ball example, introduced in Section 6.2.

As discussed in Section 6.4, we can restrict our attention to orbits of the form $\mathcal{T}\mathcal{Q}(x, h_1, h_2)$ where $x \in \tilde{S}_p$. That is, we have

$$\hat{\sigma}(\mathcal{S}) = \inf_{(x, h_1, h_2)} \max(h_1, h_2) \text{ s.t. } \mathcal{Q}(x, h_1, h_2) = x.$$

This constrained 3-dimensional nonlinear optimization problem can be reduced to the following unconstrained 2-dimensional nonlinear optimization problem:

$$\hat{\sigma}(\mathcal{S}) = \inf_{(x, h_1)} \max(h_1, {h_2}^\star(x, h_1)) \tag{6.14}$$

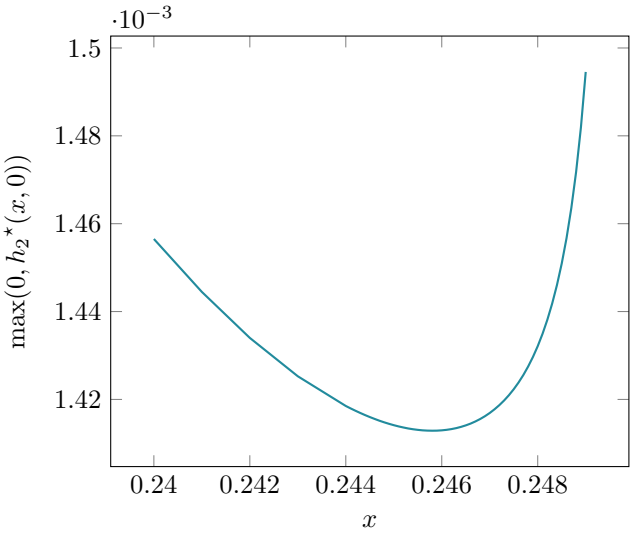


Figure 6.14: Upper bound provided by the closed orbit as a function of the point of the Poincaré curve contained in the orbit. The delay h_1 used by the orbit is 0 and the delay h_2 is ${h_2}^\star(x, 0)$.

where ${h_2}^\star(x, h_1) = \min\{\, h_2 \mid \mathcal{Q}(x, h_1, h_2) = x \,\}$. This reduction is possible because, given fixed values of x and h_1, the value of ${h_2}^\star(x, h_1)$ is straightforward to compute. Indeed, let $y \in G$ be such that $P_{S_p}^{f_1}(y) = x$, we have

$$ {h_2}^\star(x, h_1) = \tau_{\mathcal{T}}^{f_2}(x_2) - \tau_G^{f_2}(x_2) $$

where $\mathcal{T} = \mathcal{T} P_{S_p}^{f_1}(y)$ and x_2 is defined in Equation (6.8). For a given accuracy, ${h_2}^\star(x, h_1)$ can be computed using classical simulation methods for nonlinear systems (see e.g. [87]) as follows. For a given point $x \in \tilde{S}_p$, we precompute the trajectory $\mathcal{T}$ with a time steps determined by the required accuracy. Then we compute the point $\tau_{\mathcal{T}}^{f_2}(x_2)$ by first simulating the trajectory with a coarse time step. Let t, z be the last element of the sampled trajectory before the intersection with $\mathcal{T}$. We have $\tau_{\mathcal{T}}^{f_2}(x_2) = t + \tau_{\mathcal{T}}^{f_2}(z)$ and $\tau_{\mathcal{T}}^{f_2}(z)$ is smaller than the time step used to simulate the trajectory starting at x_2. We can therefore estimate $\tau_{\mathcal{T}}^{f_2}(z)$ with a refined time step. This procedure can be applied recursively. This recursion is rather cheap computationaly but increasing the number of recursion layers is not sufficient as at some point, the accuracy will be limited by the time step used for computing the trajectory $\mathcal{T}$.

In the example introduced in Section 6.2, the optimal solution of the problem (6.14) with accuracy 10^{-9} is given at $x^\star \approx 0.24579453$, ${h_1}^\star = 0$ and $h_2 = {h_2}^\star(x^\star, 0) \approx 0.0014128697$. We illustrate the objective function along the line $h_1 = 0$ in Figure 6.14. We used the library described in [311] to simulate the nonlinear system.

6.5.1 Comparison with State of the Art

Reachability techniques can be used to approximate $\sigma(\mathcal{S})$ by trial and error. For example, replacing the non-linear drag term in the bouncing ball example ($-d_a/m x_2 \, |x_2|$

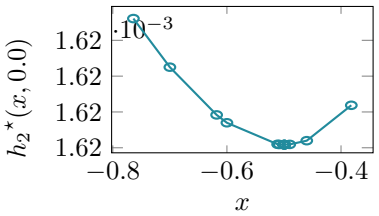


Figure 6.15: Affine bouncing ball model results. Plot of $h_2{}^\star(x, h_1)$ as a function of the point of the Poincaré curve, and $h_1 = 0$ (recall Equation (6.14)).

in Equations (6.1) and (6.2)) by a linear term $-d_a/m$ with $d_a = 10^{-6}$, allows us to use SpaceEx [129] with the model depicted in Figure 6.16, which is the same as $\mathcal{S}_H$ except the equations are affine. Applying our algorithm to the affine bouncing ball model with $h_1 = 0$, yields the values for h_2 as plotted in Figure 6.15. The minimum is $h_2{}^\star(x, 0.0) = 0.001623205$ at $x = -0.4991045$. To check these results, we ran SpaceEx with the initial conditions $x = -0.4991045, v = 0.0$ and parameters $h_1 = 0, 0 < h_2 < 0.001623205$. A fixed point was not reached, but the intermediate state space, shown in Figure 6.17, hints at the correctness of our result: the trajectories do not reach $x < -0.4991045$. Compared to our algorithm, this trial and error process to find the solution to Equation (6.14) is not feasible in practice, but can be used to check the solution found.

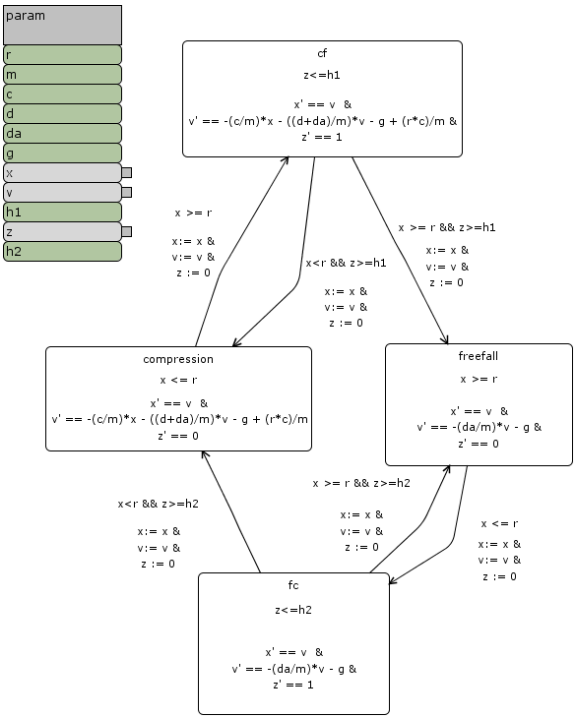


Figure 6.16: Delayed transition SpaceEx affine model of the bouncing ball example.

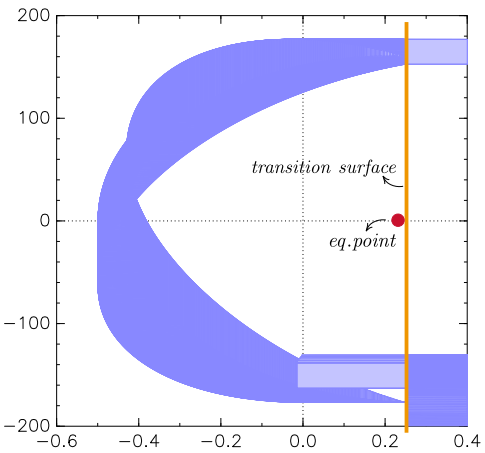


Figure 6.17: Clipped state space output, produced with SpaceEx.

6.6 Related Work

We have collected related works from three fields: stabilization of Networked Control Systems (NCS), simulation of colliding bodies, and co-simulation of hybrid systems. Compared to our work, most research works focus on linear (or affine) systems while our approach, although restricted to planar system, is developed in the general setting of nonlinear systems.

In the domain of network control, we found only one paper ([199]) that considered explicitly the delays in the switching signal. In [199], the authors focus on the state-feedback stabilization of LTI systems that can be remotely controlled over a multi-hop network. In contrast, we focus on non-linear *planar* systems with one switching surface. Their approach uses Joint Spectral Radius to prove stability, which is undecidable in general (recall Chapter 5).

The works in [36, 113, 174, 175, 177, 381] consider the stabilization of a NCS under delayed sampling and actuation signals, communication constraints, quantization, and packet dropouts. Our work focuses solely on the delays regarding the mode switching signal. Depending on the dynamics of each mode, it can be possible to encode the delayed counterpart of a BMS (Definition 30) as a NCS where the delayed switch between modes is emulated by the delayed input to the plant in the NCS. However, this is not in general possible for non-linear dynamics.

In the domain of animation of colliding multi-body systems, the focus is on detecting that collisions have occurred without allowing too much interpenetration of objects (see, e.g., [96, 111, 259, 347]). This is usually attained by relying on step-size control algorithms and backtracking. Our work might be applied to determine the simulation step size that ensures stable simulation results, without relying on expensive backtracking and correction techniques.

Regarding the approach followed, we highlight [36, 145].

In [36], the delayed system is formulated as a hybrid system. Then, the stability of which

is proven if a Lyapunov function can be found, using the sum of squares [290] approach. This allows to cover polynomial hybrid systems. Note that their approach only provides a procedure to approximate the optimal value of Problem 2.

The work in [145] provides a comprehensive study of the dynamics of planar systems with the form $\dot{x} = Ax + sign(w^T x)v$, with A a constant matrix. Moreover, Poincaré maps are also used to analyze the stability of periodic motions ([145, Section 5.2]).

6.7 Concluding Remarks

Motivated by practical problems in the numerical simulation of Hybrid Systems, we have studied how a delay in the detection of mode transitions can impact the quality of the result of the numerical simulation. It turns out that this delay may have a crucial impact on the result, as it may turn a stable Hybrid System into an unstable behaviour in the numerics.

Our goal was to provide a fundamental study of this phenomenon, and we have focused here on planar systems. A natural first research question aiming at understanding the problem is to characterize the threshold between stability and instability of the simulated trajectories, when the true system is stable. Already for this simple case, the results that allow us to compute the maximal allowed delay ensuring stability of the simulated trajectories are non-trivial.

We have connected the problem with classical techniques in the analysis of Dynamical Systems, such as Poincaré maps and topological arguments. However, we showed that in Hybrid Systems, more complex phenomena can occur, which make these classical tools insufficient to solve the problem. We pushed further these techniques, which allowed us, under mild assumptions, to compute this maximal delay for planar systems.

Limitations and Assumptions Regarding the generalization of these results to non-planar systems, the major obstacle is that Theorem 4 cannot be generalized: it is easy to image a system with a 3-D spiraling behavior, which is unstable, but no cycle can ever be found. If one can prove that a projection of the system can be made onto a plane, and that the hidden variables are not affected by the delay, then maybe we can apply the results in this chapter.

Most of the assumptions made constraint the placement of the Poincaré surface, and are there in order to ensure continuity of the Poincaré map, which we need to reduce the problem of stability into a problem of finding a cycle. For systems with more than two modes, it might be possible to generalize if we consider all permutations of modes where cycle can occur.

We showed that traditional reachability analysis techniques do not scale to solve this problem satisfactorily.

We hope that this work will provide a better understanding of the problem, of high importance in numerical simulation, and will lead to the estimation of the maximal allowed delay for more complex, or higher dimensional, hybrid systems than the ones studied here.

Chapter 7

Semantic Adaptation

Disclaimer The content in this chapter is adapted from:

- GOMES, CLÁUDIO, Bart Meyers, Joachim Denil, Casper Thule, Kenneth Lausdahl, Hans Vangheluwe, and Paul De Meulenaere. "Semantic Adaptation for FMI Co-Simulation with Hierarchical Simulators." SIMULATION, 2018, 1–29. https://doi.org/10.1177/0037549718759775.

The previous two chapters have focused on fundamental challenges in co-simulation, and have shown the impact of the configuration of each simulator in the co-simulation. In this chapter, motivated by practical issues, we develop an approach to change the configuration of each simulator, without having to ask its original developer.

7.1 Introduction

The FMI standard provides a common interface to allow a uniform communication with the black boxes, solving the combinatorial explosion of import/export formats (recall Section 3.7). However, it does not ensure that the black boxes are interacted with in a semantically correct manner, and that their implementation is satisfactory.

These aspects are important because:

- as shown in Chapter 5, and later in Chapter 8, different implementations of an FMU can lead to fundamentally different co-simulation results (.e.g, failure to preserve stability); and
- as shown in Chapter 3, the accuracy of a co-simulation is often controlled by the input approximation schemes of the FMUs, which means that a single badly implemented FMU can make the co-simulation results worthless.

When an FMU does not behave as it is expected to, we say that there is an interaction mismatch between the FMU and the co-simulation scenario. Prior work [59, 61, 110, 251, 267] suggests that these mismatches can be roughly classified as:

Signal Data Mismatch happens when the signals provided by the FMU are not compatible with the ones that are expected (e.g., different frame of reference, different physical

units).

Model of Computation Mismatch happens when the provided FMU assumes a different model of computation [231] than the one actually used to compute the overall behavior of the system (e.g., FMUs exported by a timed automata modelling and simulation tool [288, 353] have to make assumptions about the other interacting FMUs).

Capability Mismatch happens when a given FMU lacks some capabilities (recall the capabilities identified in Chapter 3) that affect the simulation performance (e.g., FMUs that lack higher order input extrapolation, an important capability that affects the accuracy and stability of the co-simulation [24, 71]).

Rather than requiring the original producer of the FMU to correct an interaction mismatch, it can be useful that the team is able to correct it immediately. Moreover, as we later show in Chapter 8, it is not always clear what the correct implementation of the FMU should be. This means that mismatches can happen between a given FMU and a usage intent, and therefore it is not necessarily the case that the best correction of a mismatch is done by the producer, if the FMU is to be reused. In fact, as [24, 35, 153, 155] and Chapter 8 show, some mismatches occur because of the way the FMUs are coupled, and the correction has to be done for that specific co-simulation scenario.

Informally, we call semantic adaptation of an FMU to the set of modifications made to the inputs/outputs and interaction with environment, of the FMU, with the purpose of correcting an interaction mismatch.

The above arguments motivate the need for semantic adaptations, and lead to the following research question:

RQ1 How can we *describe* the most common semantic adaptations on multiple types of black box FMUs in a *productive* manner, and realise them without violating *modularity* and *transparency*.

Productivity is related to the effort required to describe an adaptation. *Modularity* refers to the fact that any FMU should be adapted by changing how it is interacted with, and not how it is implemented. *Transparency* means that any tool that imports FMUs should not have to be changed in order to import, and interact with, an adapted FMU.

The descriptions should be made in an independently developed language because it is impractical that every tool capable of importing FMUs is able to implement the adaptations. Furthermore, one cannot expect that any user of an FMU has the ability to modify the importing tool to support these. Compared to implementing these adaptations manually, a language reduces the accidental complexity, prevents mistakes, and allows soundness analyses to be carried out.

In this chapter, we build on prior work [110, 251, 358] to define a language that allows for the descriptions of the most common semantic adaptations that can be used in FMI co-simulation, surveyed in [157]. A distinct feature of the language proposed here is that it describes adaptations for groups of interconnected FMUs in the same way as for a single FMU, thanks to a sound definition of hierarchical co-simulation.

The definition of hierarchical co-simulation, and the semantics of the language, are presented in a bottom up approach, as illustrated in Figure 7.1. In Section 7.2, we introduce a co-simulation abstraction with simulation units and how these relate to FMUs. Section 7.4

contains the formal foundations of a special kind of simulation unit that is the template to implement any semantic adaptation. In Section 7.3, a running example is described, and in Section 7.5 the language and its semantics are described. Section 7.6 judges how well we have addressed the research question. Section 7.7 discusses the flaws of our approach and research opportunities. Finally, Sections 7.8 and 7.9 present the related work and conclude this chapter, respectively.

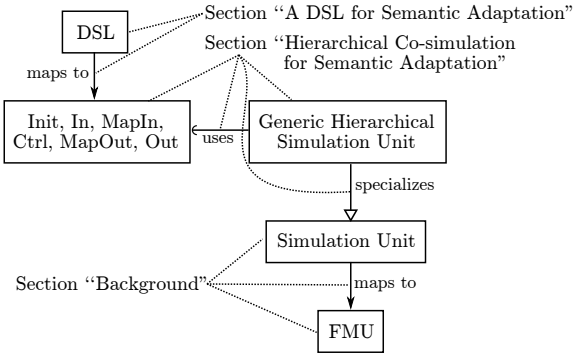


Figure 7.1: Overview of DSL semantics and chapter structure.

7.2 Background

In this section, we introduce the concepts, terminology, and assumptions used throughout this chapter. These refine the concepts introduced in Chapter 2 and Section 3.7. We cover the Functional Mockup Interface (FMI) standard, semantic adaptation, and domain specific languages.

7.2.1 Co-simulation

Recall the following concepts, described in Section 3.7: dynamical model, simulation, simulation unit, master algorithm, and co-simulation scenario.

Definition 34 (Simulation Unit)**.** We capture the essence of a simulation unit with reference i, using the discrete time system notation, in one of the following four possible ways:

$$\begin{aligned}
\left\langle \boldsymbol{x_i}(t+\tilde{H}_i), \tilde{H}_i \right\rangle &= F_i(t, H, \boldsymbol{x_i}(t), \underbrace{\boldsymbol{u_i}(t+H)}_{\text{reactive}} \text{ or } \underbrace{\boldsymbol{u_i}(t)}_{\text{delayed}}) \\
\boldsymbol{y_i}(t) &= \underbrace{G_i(t, \boldsymbol{x_i}(t), \boldsymbol{u_i}(t))}_{\text{Mealy}} \text{ or } \underbrace{G_i(t, \boldsymbol{x_i}(t))}_{\text{Moore}} \\
\boldsymbol{x_i}(0) &= \underbrace{Init_i(\boldsymbol{u_i}(0))}_{\text{reactive}} \text{ or } \underbrace{Init_i()}_{\text{delayed}},
\end{aligned} \tag{7.1}$$

where t denotes the simulated time, $\boldsymbol{x_i}$ denotes the state vector, $\boldsymbol{u_i}$ the input vector, $Init_i$ computes the initial state, $H > 0$ denotes the requested communication step size, $0 < \tilde{H}_i \leq H$ denotes the communication step size taken by the unit, F_i is the state transition function, and G_i the output function.

Bold symbols will always refer to vectors in this chapter.

Definition 34 covers the different kinds of simulation units considered (based on the master algorithms surveyed in [157]): Reactive Mealy, Reactive Moore, Delayed Mealy, and Delayed Moore. The difference is in how the unit expects all inputs to be provided. For example, a delayed Moore unit can compute its output without requiring an input, and can compute its future state ($\boldsymbol{x_i}(t + \tilde{H}_i)$) with just the current input $\boldsymbol{u_i}(t)$. A reactive Mealy unit, on the other hand: requires an initial input to compute the initial state; and needs to know the next input in order to compute the next state/output.

We use shortcuts such as $F_i(t, H, \boldsymbol{x_i}(t), \ldots)$, $G_i(t, \ldots)$, and $Init_i(\ldots)$, to denote the appropriate function depending on the kind of unit i. Furthermore, note that F_i and G_i are mathematical functions (also denoted pure).

We make the following assumptions about each simulation unit i.

Assumption 7. *The internal definition of F_i and G_i is unknown, but the kind of unit (Definition 34) is known.*

Assumption 8. *If $\left\langle \cdot, \tilde{H}_i \right\rangle = F_i(t, H, \boldsymbol{x_i}(t), \ldots)$ and $\tilde{H}_i < H$, then unit i rejects the step size H requested. Furthermore, for any $\tilde{H} \leq \tilde{H}_i$, we assume that $\left\langle \cdot, \tilde{H} \right\rangle = F_i(t, \tilde{H}, \boldsymbol{x_i}(t), \ldots)$.*

Assumption 7 reflects the fact that the FMI standard does not include information about the reactivity of the units. Therefore, the works in the state of the art make a similar assumption. Assumption 8 reflects the FMI standard protocol to detect when a step size as been rejected.

Given a set of unique unit references $D = \{1, \ldots, n\}$, a *co-simulation scenario* is defined as the aggregation of each simulation unit (Definition 34), plus a coupling function that defines the input of i as a function of the outputs of units $\{j : j \in D \setminus \{i\}\}$. Formally, combining the notation used in [24, 217], a scenario is given by:

$$\begin{cases} \left\langle \boldsymbol{x_i}(t + \tilde{H}_i), \tilde{H}_i \right\rangle = F_i(t, H, \ldots) \\ \boldsymbol{y_i}(t) = G_i(t, \ldots) \\ \boldsymbol{u_i} = c_i(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{i-1}, \boldsymbol{y}_{i+1}, \ldots, \boldsymbol{y}_n) \\ \boldsymbol{x_i}(0) = Init_i(\ldots) \\ \quad \text{for each} \quad i \in D \end{cases} \tag{7.2}$$

where c_i denotes the coupling function, and each F_i, G_i follows Definition 34. Commonly, c_i is linear and maps at most one component of one of the inputs (the inputs/outputs are vector quantities), onto one component of the output. We assume that c_i is linear.

Let $i, j \in D$ be two different units, and $\bar{0}$ be the zero matrix of appropriate dimension. If $\frac{\partial c_i}{\partial \boldsymbol{y_i}} \neq \bar{0}$, then i *gets part of its input from* j. Informally, this means that at least one component of $\boldsymbol{u_i} = c_i(\ldots)$ is determined by at least one component of $\boldsymbol{y_i}$. We say that a unit $i \in D$ *depends algebraically* on unit $j \in D$, if i gets part of its input from j and i is *not* a *delayed Moore*. So, e.g., if i gets part of its input from j, but it is a *delayed Moore*, then i does not depend algebraically on j.

Using the algebraic dependency relationship, one can build a directed graph — called the dataflow graph — with one node n_i per simulation unit $i \in D$, and an edge (n_j, n_i)

between two nodes n_j, n_i whenever the unit i depends algebraically on unit j. This procedure is based on the Causal Block Diagram Simulation algorithm [151, 304]. A topological order of the resulting graph gives an execution order that respects the units' algebraic dependencies.

Depending on the coupling function and on the kind of simulation units being coupled, *algebraic loops* may occur. An algebraic loop includes any input/output/state that depends on itself, at the same time point [217].

If an algebraic loop exists between the units, then it is not possible to compute a topological ordering of the dataflow graph. For now, to simplify the explanation, we assume that such topological order can always be computed for a given co-simulation scenario. We denote that order via a mapping $\sigma : \mathbb{N} \to D$, that returns the unit reference $\sigma(j)$ that is the j-th in the topological order. So $\sigma(1)$ gives a unit that is first in the topological order, i.e., has no algebraic dependencies. This procedure is revisited in Chapter 8.

With a well defined topological order, the master only has to provide inputs to, execute, and get outputs from, the units in that order. Algorithm 6 formalizes what is known in the state of the art as the Gauss-Seidel master. It computes the behavior trace of a given co-simulation scenario as described in Equation (7.2). To be concise, we abbreviate the output and state transition function calls, which depend on the kind of unit (lines 11, 19, and 21). Furthermore, the master provides the inputs ($\boldsymbol{uc_{\sigma(j)}}$ or $\boldsymbol{up_{\sigma(j)}}$, in line 19) that each unit expects, working for both reactive and delayed units alike.

We single out Algorithm 6 in this chapter because it supports all kinds of simulation units. It therefore forms the basis for the formalization of hierarchical co-simulation units.

Without loss of generality, we assume the most basic step size control policy in Algorithm 6: the communication step size is never increased after being rejected by some unit[1]. The master uses the most recent consistent state.

7.2.2 Functional Mock-up Interface Standard (FMI)

The FMI standard [48] defines the interface and interaction pattern that allows simulation units to communicate. In the standard, a simulation unit is called a Functional Mockup Unit (FMU).

7.2.2.1 FMUs and Simulation units

This subsection establishes the equivalence between FMUs and simulation units (recall Figure 7.1), and the assumptions we make throughout this document.

Given a simulation unit i (Definition 34) we define its equivalent FMU, and vice versa, as follows:

FMU State – The state of the FMU corresponds to the state of the unit x_i. The FMU does not make the state explicit, but instead implements functions that can be used to set and retrieve the state.

Inputs – FMUs have input ports, each accepting a scalar quantity. Each dimension in the input $\boldsymbol{u_i}$ corresponds to one input port of the FMU. The FMU implements functions that

[1] Algorithm 6 can be greatly optimized (e.g., rolling back as soon as a reject occurs).

ALGORITHM 6: Gauss-seidel master for co-simulation scenarios can be topologically sorted.

Data: The stop time T, a starting communication step size $\hat{H}$, and a set of unit references $D = \{1, \ldots, n\}$.

$t := 0$; // Simulation time
$H := \hat{H}$; // Communication step size
// Initialize variables
for $i = 1, \ldots, n$ **do**
  $\boldsymbol{x_i} := \mathbf{0}$; // State vector
  $\boldsymbol{uc_i} := \boldsymbol{y_i} := \mathbf{0}$; // Current I/O variables
  $\boldsymbol{up_i} := \mathbf{0}$; // Previous input variables
end
// Compute initial states
for $j = 1, \ldots, n$ **do**
  $\boldsymbol{uc_{\sigma(j)}} := c_{\sigma(j)}(\boldsymbol{y_1}, \ldots, \boldsymbol{y_{\sigma(j)-1}}, \boldsymbol{y_{\sigma(j)+1}}, \ldots, \boldsymbol{y_n})$;
  $\boldsymbol{x_{\sigma(j)}} := Init_{\sigma(j)}(\boldsymbol{uc_{\sigma(j)}})$ or $Init_{\sigma(j)}()$
  $\boldsymbol{y_{\sigma(j)}} := G_{\sigma(j)}(t, \boldsymbol{x_{\sigma(j)}}, \boldsymbol{uc_{\sigma(j)}})$ or $G_{\sigma(j)}(t, \boldsymbol{x_{\sigma(j)}})$;
  $\boldsymbol{up_{\sigma(j)}} := \boldsymbol{uc_{\sigma(j)}}$;
end
while $t < T$ **do**
  $accepted := false$;
  while $not\ accepted$ **do**
    for $j \in (1, \ldots, n)$ **do**
      $\boldsymbol{uc_{\sigma(j)}} := c_{\sigma(j)}(\boldsymbol{y_1}, \ldots, \boldsymbol{y_{\sigma(j)-1}}, \boldsymbol{y_{\sigma(j)+1}}, \ldots, \boldsymbol{y_n})$;
      $\left\langle \tilde{\boldsymbol{x}}_{\boldsymbol{\sigma(j)}}, \tilde{H}_{\sigma(j)} \right\rangle := F_{\sigma(j)}(t, H, \boldsymbol{x_{\sigma(j)}}, \boldsymbol{uc_{\sigma(j)}}$ or $\boldsymbol{up_{\sigma(j)}})$;
      $\boldsymbol{y_{\sigma(j)}} := G_{\sigma(j)}(t + \tilde{H}_{\sigma(j)}, \boldsymbol{x_{\sigma(j)}}, \boldsymbol{uc_{\sigma(j)}})$
        or $G_{\sigma(j)}(t + \tilde{H}_{\sigma(j)}, \boldsymbol{x_{\sigma(j)}})$;
    end
    $\tilde{H} := \min_{i \in D}(\tilde{H}_i)$;
    if $\tilde{H} < H$ **then**
      $H := \tilde{H}$;
    else
      $accepted := true$;
    end
  end
  // Commit state and update I/O
  for $j = 1, \ldots, n$ **do**
    $\boldsymbol{x_i} := \tilde{\boldsymbol{x}}_{\boldsymbol{i}}$;
    $\boldsymbol{up_i} := \boldsymbol{uc_i}$;
  end
  $t := t + H$; // Advance time
end

allow the master to set those inputs (e.g., `fmi2SetReal` and `fmi2SetInteger`) and a single vector quantity $\boldsymbol{u_i}$ can be set via multiple calls to those functions.

Outputs – The outputs of the FMU are analogous to the inputs. To obtain an output $\boldsymbol{y_i}$, multiple calls are made to the dedicated functions (e.g., `fmi2GetReal` and `fmi2GetInteger`).

Initial State – The initial state computed by the $Init_i$ function corresponds to the computation performed by the FMU in the initialization mode.

Co-simulation Step – A state transition invocation $\left\langle \tilde{\boldsymbol{x}}_{\boldsymbol{i}}, \tilde{H}_i \right\rangle := F_i(t, H, \boldsymbol{x_i}, \boldsymbol{u_i})$ is mapped to (in order): an optional invocation to set the state of the FMU to $\boldsymbol{x_i}$; multiple invocations to set the input $\boldsymbol{u_i}$; an invocation to the `fmi2DoStep` function; a query to find out up to which time the FMU computed the step (to get $\tilde{H}_i$); and an (optional) invocation to get the new state of the FMU $\tilde{\boldsymbol{x}}_{\boldsymbol{i}}$. The manipulation of the state is optional for master algorithms that do not perform rollback operations.

Output Function – If the unit is a Mealy unit, then the execution of the output function $\boldsymbol{y_i} := G_i(t, \boldsymbol{x_i}(t), \boldsymbol{u_i}(t))$ corresponds to setting the inputs to the FMU, and then

getting the outputs. If the unit is a Moore unit, then the outputs can be enquired without first setting the inputs.

Assumption 9. *We assume that an initial state of a unit/FMU can always be found from the $Init(\ldots)$ function (and initial input, in case of a reactive unit).*

Assumption 10. *Every FMU supports rollback.*

Assumption 9 is in accordance with the FMI Standard, but it *restricts our scope* to scenarios were the consistent initial state of one unit depends on factors (e.g., the initial state of another unit) other than its initial inputs.

It is the role of the master to set the appropriate inputs depending on whether the FMU is reactive or delayed, or Mealy and Moore.

Assumption 11. *We define the type of the FMU by applying the following rules, in order:*

1. *If the unit does not disclose any input-to-output feed-through, it is assumed to be Mealy.*
2. *If at least one output variable depends instantaneously on an input variable, we assume that the unit is Mealy.*
3. *If the previous two do not apply, the unit is assumed to be Moore.*
4. *If the capability flag* `canInterpolateInputs` *is set, then the unit is* reactive*.*
5. *Otherwise, the unit is delayed.*

To establish the equivalence of the couplings restrictions of units and those of FMUs, we note that the definition of algebraic dependency remains the same between FMUs. Thus, the dataflow graph can be built as described in the previous subsection.

Having established the equivalence between simulation units and FMUs, we will henceforth use the two terms interchangeably.

7.2.3 Semantic Adaptation

The interface of an FMU (or of a simulation unit) comprises not only the specification of the inputs and outputs, but also how it is to be interacted with [251]. It may be the case that in different co-simulation scenarios, the same FMU has to be interacted with differently (e.g., for accuracy/performance concerns). While modifying the master to support a new interaction pattern will solve the problem, it is not ideal since: (i) the interaction pattern may be specific to a single FMU (therefore not reusable), and (ii) modifications to the master may require extensive testing to ensure that it retains its correctness properties (e.g., see [143]).

Our work avoids changes to the underlying master algorithm, and focuses those changes around the FMU itself in the form of semantic adaptations, using hierarchical co-simulation.

An adaptation targets an FMU, or group of FMUs, which we will call the *internal FMU(s)*, and the end result of an adaptation is a new FMU, which we call *external FMU*. The external FMU interacts with the internal FMU(s), without requiring them to be modified (*modularity*). The adjectives *internal* and *external* reflect the hierarchical nature of our approach and are illustrated in Figure 7.2.

We introduce below a non-exhaustive list of semantic adaptations that can be classified according to the interaction mismatch they intent to correct:

Signal Data Mismatch: Conversion of Units and Reference Frame translation.

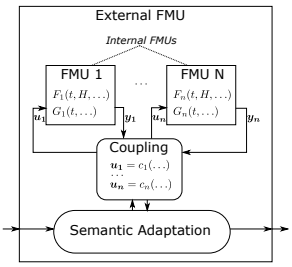


Figure 7.2: Internal FMUs, External FMU, and Semantic Adaptation.

Model of Computation Mismatch: Hold, Quantization, Data Triggered Execution, and Timed Transitions.

Capability Mismatch: Interpolation/Extrapolation of Inputs, Fixed Point Iteration, Multi-Rate Adaptation, Time and Partial Derivative Adaptation, Accurate Threshold Crossing, and Re-Initialisation.

See [150, 157] and references thereof, for variants of these adaptations.

7.2.3.1 Conversion of Units and Reference Frame Translation

The conversion of units and reference frame adaptations, take an internal FMU and create an external FMU whose inputs/outputs are algebraic transformations of the input/outputs of the internal FMU.

7.2.3.2 Interpolation/Extrapolation of Inputs

An FMU that stands for a continuous system, such as a DC motor, approximates its behavior trace by discretizing the time continuum into a finite set of points [87] and applying a numerical method at each of those points.

In co-simulation, when the master asks an FMU to compute the behavior trace over an interval of time, from t to $t + H$, the FMU discretizes the interval and computes the internal solution at each of these points, called micro-steps. The most common FMUs assume that, in between t and $t + H$ the inputs provided by the master are constant. Naturally, for large H, this assumption causes a significant error in the co-simulation [19, 23, 71, 329].

Instead of reducing H, it is possible to adapt the FMU to better approximate its inputs. Essentially, the external FMU discretizes the interval $t \rightarrow t + H$ and runs the state transition function of the internal FMU multiple times, providing an approximated input at each of the time points. The internal FMU will still assume a constant input, but will do so in smaller intervals of time.

7.2.3.3 Fixed Point Iteration

If an algebraic loop exists, then the involved units will belong to the same cycle in the corresponding dependency graph.

As proposed in [150, 358], given a co-simulation scenario (recall Equation (7.2)) that has one cycle involving at least two simulation units (non-trivial), it is possible to create an

external FMU that replaces all the units in the cycle. All the couplings external to the cycle become couplings to the hierarchical simulation unit.

At each state transition of the external FMU, a fixed point iteration technique is applied to the inputs/outputs of the internal FMUs.

If a scenario has multiple non-trivial cycles, this adaptation can be applied to reduce the scenario to one where all the algebraic loops are solved [358]. Algorithm 6 can then be applied to compute the co-simulation.

7.2.3.4 Multi-Rate Adaptation

For FMUs simulating first order Ordinary Differential Equations (ODE), the larger the interval between the points, the less accurate the computed behavior trace will be [69].

The multi-rate adaptation is used to increase the accuracy while not sacrificing the performance in a co-simulation. Applied to co-simulation, the technique, well known in the circuit simulation domain [235], consists of a groups of interconnected internal FMUs that communicate more frequently [162, 358]. This can serve two purposes: optimize the communication cost between the internal units [150], or optimize the accuracy of the co-simulation (especially when the internal units are physically tightly coupled [157]).

Similarly to the input extrapolation adaptation, the state transition function of the external unit instructs the internal units to perform multiple steps and exchange values at each of those steps. The higher the rate of the adaptation, the higher the number of internal steps performed.

This adaptation can be combined with the approximation of inputs adaptation, to provide for approximated inputs at each of the internal state transition invocations.

7.2.3.5 Time and Partial Derivative Adaptation

Time and partial derivative information about each simulation unit's outputs can be used to optimize the co-simulation process in many different ways (e.g., see [332]).

In the FMI standard, since the FMUs can optionally provide time and partial derivative information, it is often the case that some units do not support it. To mitigate this, a derivative adaptation can be used to produce an external FMU that provides (numerically estimated) partial and time derivatives.

7.2.3.6 Accurate Threshold Crossing

A co-simulation trace is more accurate if all units exchange values at the time when a certain signal crosses a given threshold. The problem of accurately finding that time is well known in the hybrid system simulation domain [260, 388] and many techniques exist to address it [69, 87]. In FMI co-simulation, the most basic technique to accurately locate a crossing consists of rejecting a step size and proposing a new one, that possibly coincides with the threshold crossing moment.

The accurate zero crossing adaptation ensures that the external FMU rejects the proposed step size when one of the inputs of the internal FMU crosses a significant threshold *too late* [110].

7.2.3.7 Re-Initialisation

An internal FMU that is expecting a smooth input signal may yield unexpected behavior trace when given a discontinuous signal (we consider a discontinuous signal to be a sufficiently rapid changing one in between co-simulation communication points) [70, 71, 116]. For example, an FMU that is using a multi-step numerical solver which assumes the input to be continuous (see, e.g., [20] for a possible solution to this problem).

A re-initialization adaptation ensures that the external unit: (1) locates accurately the time of the discontinuity (e.g., in the same manner as the accurate crossing adaptation), and (2) the external unit is properly reset before handling the new value of the input. In the FMI standard, item (2) requires three steps: save the unit state; reset and initialize the unit; and restore the state.

7.2.3.8 Quantization

Quantization is an adaptation commonly used to convert a continuous signal into a discrete event one. The (continuous) set of possible input values is discretized into regions and, during the co-simulation, whenever the continuous signal enters a new region, an event is produced [53, 211].

In co-simulation, this adaptation transforms an internal FMU that expects continuous inputs and produces continuous outputs, into an external FMU that deals with events (see, e.g., [31, 54, 79, 310]).

The realization of this adaptation is very similar to the zero crossing one, except that the thresholds to locate are induced by the input space discretization.

7.2.3.9 Hold

The hold family of adaptations can be seen as the dual of the multi-rate adaptations.

If an internal FMU should run slower than the rest of the simulation units, then it can be adapted with a hold adaptation. The external FMU will *trick* the master and obey to the proposed step sizes, but will avoid executing the internal FMU every time a step is requested. For example, if a zero order Hold adaptation is used, then the external unit will produce an output that is equal to the most recent output produced by the internal unit.

There are many variants of this adaptation, with varying degrees of accuracy, borrowed from well known approximation techniques [69].

The two adaptations below are novel in FMI based co-simulation domain, but well known in the discrete event domain.

7.2.3.10 Data Triggered Execution

The data triggered execution is an adaptation most useful when the modeller knows that a particular internal FMU will only produce relevant behavior when certain conditions are true over its inputs. The adaptation executes the internal FMU only when these conditions are met.

7.2.3.11 Timed Transitions

The time transition adaptation can be used when the internal FMU is known to have internal state changes, triggered after a known amount of time. The adaptation will query the internal FMU to know when exactly should the next state transition function call take place, and will call it only when that time is arrived. It can be combined with the data triggered execution to achieve a *lazy* execution of units.

Each of the semantic adaptations described above has many variants that make its ad-hoc implementations not only error prone, but also tedious. Additionally, one can extract the shared commonalities in the implementation of all semantic adaptations. The interplay between many small variants and shared commonalities is one of the motivating factors to use a Domain Specific Language for the description of the adaptations.

7.2.4 Domain-Specific Languages

Domain-specific languages (DSLs) offer a way to deal with the essential complexity of a given domain, while avoiding its accidental complexity [205].

We highlight two important advantages that come from the use of a DSL in the context of our contribution:

1. The most common tasks in the target domain are performed in a very simple, productive, and intuitive manner (for a trained domain expert) — the descriptions made in our DSL do not deal with the idiosyncrasies of an implementation of the FMI Standard, even though a FMI compliant external FMU can be generated.
2. By maximally constraining the user, a DSL ensures that he/she makes less mistakes and allows domain level validation — our DSL allows the user to specify extra information that can be used to detect mistakes (a simple validation being the compatibility of units in inputs/outputs).

7.3 Running Example

To showcase the language, the case study we present is adapted from a power window system, described in [107, 307]. This system is the familiar automated car window, which responds to the driver/passenger pressing up/down buttons to raise/lower it. If an obstruction is detected, the window retracts for a few moments to avoid injury. This example was chosen for its heterogeneity and need for semantic adaptations, described later.

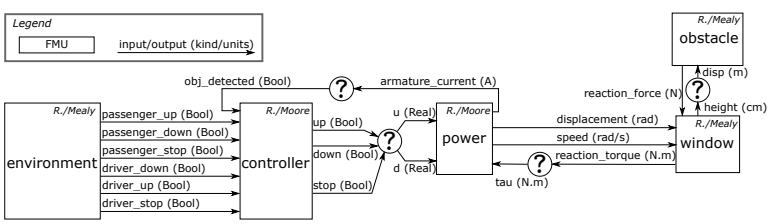


Figure 7.3: Power window co-simulation scenario.

7.3.1 The Example Scenario

Figure 7.3 shows the co-simulation scenario of the power window, consisting of five FMUs, with the illustrated input and output ports. The figure is a block representation of a co-simulation scenario as described in Equation (7.2). The FMUs were produced by the authors using independent tools.

The environment FMU, coded manually, is an abstraction of the behavior of the driver and passenger. Whenever the driver/passenger pushes a button up/down, the respective output will *pulse* to signal the event. When the button is released, the stop output pulses.

The controller FMU, produced from the Yakindu Statecharts tool, represents the software subsystem that ensures the safe operation of the window. It gets boolean pulse inputs and decides whether the motor should go up or down, through its boolean pulse outputs. If an object is detected (that is, obj_detected pulses) and the passenger (or driver) has pushed the up button, then the controller should instruct the DC motor to go down for one second. This is done by pulsing the down output and, after 1 second, pulsing the stop output.

The power is an ODE based unit, exported with OpenModelica, representing the DC motor and the up/down switched circuit that drives the motor. Whenever the u input is bigger than 0.5, the DC motor moves the window up. Analogously, whenever the d input is bigger than 0.5, it moves the window down.

The window and obstacle are stateless units, coded manually, that map the inputs to the outputs using algebraic equations. The obstacle FMU outputs a force proportional to how compressed it is. Non-zero compression happens only when the input displacement exceeds a given threshold (e.g., $0.45m$).

An object is detected when the armature_current spikes, caused by a sudden increase in the reaction_torque input of the DC motor, cause in turn by an increase in the reaction_forced of the object being compressed.

As illustrated in the figure, *all units in this example are reactive*, so the controller, power, window, and obstacle form a single cycle. The power and controller are *Moore* and the remaining units are *Mealy*.

Figure 7.4 shows the behavior trace of the example produced via a monolithic model produced in OpenModelica [133]. In the figure, the driver continuously pushes the up button, asking the controller to move the window up, but the controller detects an object at about 2.5 seconds (due to the armature current spike), which causes it to override the requests of the driver and retract the window for 1 second.

7.3.2 Semantic Adaptations

The scenario presented in Figure 7.3 cannot be used *as is* to compute a co-simulation as the one shown in Figure 7.4 because the FMUs are incompatible.

The adaptations that need to be made were introduced in the "Background" section, and are detailed in the list below and illustrated in Figure 7.5. These will be referred to throughout this document.

lazy_sa – for controller:

- execute only if the inputs change (*data triggered execution*).

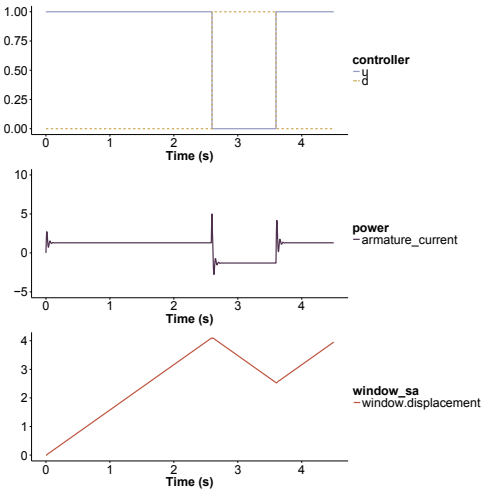


Figure 7.4: Power window monolithic simulation results.

- execute only when its state transition needs to be called (*timed transition adaptation*) due to internal triggers. In FMI, this information can be obtained by asking controller to perform a very large step.
- zero order hold its outputs.

controller_sa – for lazy_sa:

- map the armature_current to a boolean signal object_detected that is $true$ whenever there is a threshold crossing. The condition that defines the crossing is $|\text{armature\_current}| > 5$[2] and the lazy_sa unit should be invoked at the time of crossing.
- convert output, taking into account the stop signal.

window_sa – for window:

- negate the reaction_torque value;
- convert the units of height from centimetres to metres.

power_sa – for power:

- ignore the algebraic loop between controller and power, and between the power and window, by delaying the outputs of the power by one co-simulation step. This effectively makes the external FMU a delayed unit.

loop_sa – for window_sa and obstacle:

- solve the algebraic loop between obstacle and window_sa by successive substitution providing an initial guess for height.

rate_sa – in order to prevent divergence in the fixed point iteration caused by the above adaptation, smaller communication step sizes should be taken between the obstacle and the window FMUs. To this end:

- use a multi-rate adaptation, where loop_sa is executed 10 times faster than the remaining scenario.
- interpolate the input signal motor_speed.

[2]The value 5 is used here for the purposes of illustration. In practice, it is obtained by calibration with the DC Motor.

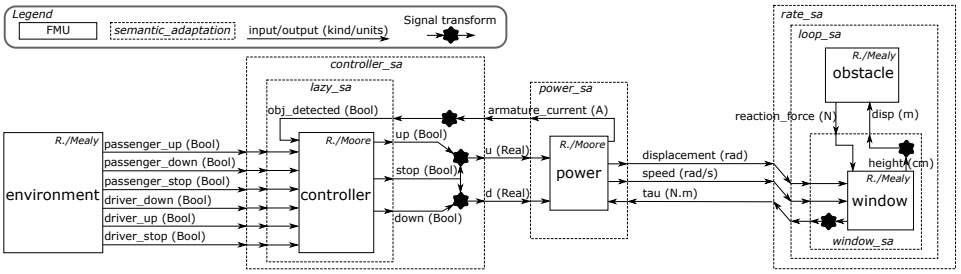


Figure 7.5: The modelled adaptations in the power window example.

7.4 Hierarchical Co-simulation for Semantic Adaptation

The most straightforward way of dealing with semantic adaptations is by creating a master algorithm that implements them. There are multiple problems with this approach: 1) it forces the master algorithm to be specific to the scenario, which hinders the potential for reuse; and 2) it violates the *transparency* principle by not allowing the FMU (plus adaptations) to be easily imported onto other tools that perform co-simulation, such as Simulink®, INTO-CPS [226], or DACCOSIM [136].

To avoid these problems, we implement the semantic adaptations as FMUs, in a hierarchical way. In fact, our language defines semantic adaptations (plus internal FMUs) as FMUs themselves, allowing for adaptations to be described "on top of" other adaptations. This way, the master and semantic adaptations can be clearly separated, as well as the semantic adaptations between themselves.

As part of our contribution, we extend the definitions provided in Section 7.2 to explain what hierarchical co-simulation is, and we give an overview on how the main semantic adaptations are implemented.

7.4.1 Hierarchical Co-simulation

Before giving the formal definition of hierarchical co-simulation, we start with an example of a "default" hierarchical co-simulation unit is and does.

A default hierarchical simulation unit is one that *wraps* a set of connected internal units, along with their inter-dependencies, and behaves in a manner that is indistinguishable from any other simulation unit. The internal FMUs have internal inputs/outputs (in between the units) and external inputs/outputs. This is called the default hierarchical unit because it does not adapt the behavior of the internal units. It merely wraps them.

To give details about how the default hierarchical unit is constructed, we extend the definition of co-simulation scenario to make the distinction between internal and external inputs. Let $\boldsymbol{u_{ext}}$ denote the input vector that is external to the co-simulation scenario. A co-simulation scenario with $D = \{1, \ldots, n\}$ units, and with external input $\boldsymbol{u_{ext}}$, is then

described as:

$$\begin{cases} \left\langle \boldsymbol{x_i}(t+\tilde{H}_i), \tilde{H}_i \right\rangle = F_i(t, H, \ldots) \\ \boldsymbol{y_i}(t) = G_i(t, \ldots) \\ \boldsymbol{u_i}(t) = c_i(\boldsymbol{u_{ext}}(t), \boldsymbol{y_1}(t), \ldots, \boldsymbol{y_{i-1}}(t), \boldsymbol{y_{i+1}}(t), \ldots, \boldsymbol{y_n}(t)) \\ \boldsymbol{x_i}(0) = Init(\ldots) \\ \text{for each} \quad i \in D \end{cases} \tag{7.3}$$

Given then a co-simulation scenario as defined in Equation (7.3), and assuming that the topological order $\sigma : \mathbb{N} \to D$ is well defined, the default hierarchical *reactive Mealy* FMU is constructed by:

1. aggregating the state $\boldsymbol{x_i}$ and the previous input $\boldsymbol{up_i}$ of each FMU i, into a single entity $\boldsymbol{x}$ that becomes the state of the hierarchical unit;
2. implementing the state transition function as a single co-simulation step of Algorithm 6.

Formally, the unit is defined as:

$$\begin{aligned} \left\langle \boldsymbol{x}(t+\tilde{H}), \tilde{H} \right\rangle &= F(t, H, \boldsymbol{x}(t), \boldsymbol{u_{ext}}(t+H)) \\ \boldsymbol{y}(t) &= G(t, \boldsymbol{x}(t), \boldsymbol{u_{ext}}(t)) \\ \boldsymbol{x}(0) &= Init(\boldsymbol{u_{ext}}(0)) \end{aligned} \tag{7.4}$$

where: $\boldsymbol{x} = [\boldsymbol{up_1}, \ldots, \boldsymbol{up_n}, \boldsymbol{x_1}, \ldots, \boldsymbol{x_n}]^T$ is the total state vector and $[\cdot]^T$ is the matrix transpose operation; the initial state vector is calculated by the $Init$ function, defined in Algorithm 7, which finds the initial inputs and states to each of the internal units depending on their types; $\boldsymbol{u_{ext}}$ is the external input vector; function G is described in Algorithm 8, which computes the outputs of all internal units from the given inputs; and function F is detailed in Algorithm 9, which executes a single co-simulation step of Algorithm 6 and returns the minimum step size selected.

The construction of the default hierarchical reactive Moore, delayed Mealy, or delayed Moore, is done similarly and we omit it. The next subsection presents similar constructions for all kinds of units, incorporating adaptations.

The default hierarchical unit gives the basic transformation that underlies the semantic adaptation of one, or a connected group of, internal FMUs. In the subsection below, we describe the generic mechanism that enables the creation of hierarchical units with semantic adaptations.

7.4.2 Generic Semantic Adaptation

Previous work [110, 251] supports the hypothesis that any semantic adaptation can be described by the following elements, that mediate the interactions of the external FMU with the internal units:

- *external input rules*, describing how the inputs provided to the external FMU are stored in the state of the external FMU;

ALGORITHM 7: *Init* function of the default hierarchical *reactive Mealy*, described in Equation (7.4). This function initializes the child units according to their type (recall Definition 34). The input to the initialization of each child unit may depend on the outputs of other units after their initialization, hence this algorithm needs to propagate outputs to inputs.

Function $Init(\boldsymbol{u}_{ext})$
  for $i = 1, \ldots, n$ **do**
    $\boldsymbol{x_i} := \boldsymbol{up_i} := \boldsymbol{y_i} := \boldsymbol{0}$;
  end
  for $j \in (1, \ldots, n)$ **do**
    $\boldsymbol{up_{\sigma(j)}} := c_{\sigma(j)}(\boldsymbol{u_{ext}}, \boldsymbol{y_1}, \ldots, \boldsymbol{y_{\sigma(j)-1}}, \boldsymbol{y_{\sigma(j)+1}}, \ldots, \boldsymbol{y_n})$;
    $\boldsymbol{x_{\sigma(j)}} := Init_{\sigma(j)}(\boldsymbol{up_{\sigma(j)}})$ or $Init_{\sigma(j)}()$;
    $\boldsymbol{y_{\sigma(j)}} := G_{\sigma(j)}(0, \boldsymbol{x_{\sigma(j)}}, \boldsymbol{up_{\sigma(j)}})$
      or $G_{\sigma(j)}(0, \boldsymbol{x_{\sigma(j)}})$;
  end
  return $[\boldsymbol{up_1}, \ldots, \boldsymbol{up_n}, \boldsymbol{x_1}, \ldots, \boldsymbol{x_n}]^T$;
end

ALGORITHM 8: Output function of the default hierarchical *reactive Mealy*, described in Equation (7.4). This function needs to exchange data of the child units before compute the output of the hierarchical unit.

Function $G(t, [\boldsymbol{up_1}, \ldots, \boldsymbol{up_n}, \boldsymbol{x_1}, \ldots, \boldsymbol{x_n}]^T, \boldsymbol{u_{ext}})$
  for $i = 1, \ldots, n$ **do**
    $\boldsymbol{uc_i} := \boldsymbol{y_i} := \boldsymbol{0}$;
  end
  for $j \in (1, \ldots, n)$ **do**
    $\boldsymbol{uc_{\sigma(j)}} := c_{\sigma(j)}(\boldsymbol{u_{ext}}, \boldsymbol{y_1}, \ldots, \boldsymbol{y_{\sigma(j)-1}}, \boldsymbol{y_{\sigma(j)+1}}, \ldots, \boldsymbol{y_n})$;
    $\boldsymbol{y_{\sigma(j)}} := G_{\sigma(j)}(t, \boldsymbol{x_{\sigma(j)}}, \boldsymbol{uc_{\sigma(j)}})$
      or $G_{\sigma(j)}(t, \boldsymbol{x_{\sigma(j)}})$;
  end
  return $[\boldsymbol{y_1}, \ldots, \boldsymbol{y_n}]^T$;
end

- *internal input rules*, detailing how the values stored internally are mapped into inputs of the internal FMUs;
- *control rules*, determining what happens when the state transition function of the external FMU is invoked;
- *internal output rules*, describing how the outputs of the internal FMUs are stored in the state of the external FMU;
- *external output rules*, detailing how the values stored in the state of the external FMU are mapped to output values of the external FMU;

This subsection formalizes how a generic external FMU incorporating the above rules is constructed.

To formalize the above rules, we define the state of the external FMU. The external FMU is constructed from a given co-simulation scenario, defined in Equation (7.3), with $D = \{1, \ldots, n\}$ units and external input vector $\boldsymbol{u_{ext}}$. Its state is then defined as

$$\boldsymbol{x} = [\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}, \boldsymbol{x_1}, \ldots, \boldsymbol{x_n}]^T$$

with $\boldsymbol{x_{in}}$, $\boldsymbol{x_{ctrl}}$, and $\boldsymbol{x_{out}}$, denoting the input, output and control storage vectors, respectively, and $\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$ being the internal units' states. The vectors $\boldsymbol{x_{in}}$, $\boldsymbol{x_{ctrl}}$, and $\boldsymbol{x_{out}}$ form the semantic adaptation storage and depend on the adaptations implemented in the external FMU.

ALGORITHM 9: State transition function of the default hierarchical *reactive Mealy*, described in Equation (7.4). This function needs to exchange data of the child units, and step them, to ensure that every child unit is at time $t + H$. If a unit rejects the step size, then the hierarchical unit needs to informs its environment of the smallest step size that could be taken.

Function $F(t, H, [\boldsymbol{up_1}, \ldots, \boldsymbol{up_n}, \boldsymbol{x_1}, \ldots, \boldsymbol{x_n}]^T, \boldsymbol{u_{ext}})$
  for $i = 1, \ldots, n$ **do**
    $\boldsymbol{uc_i} := \boldsymbol{y_i} := \boldsymbol{0}$;
  end
  for $j \in (1, \ldots, n)$ **do**
    $\boldsymbol{uc_{\sigma(j)}} := c_{\sigma(j)}(\boldsymbol{u_{ext}}, \boldsymbol{y_1}, \ldots, \boldsymbol{y_{\sigma(j)-1}}, \boldsymbol{y_{\sigma(j)+1}}, \ldots, \boldsymbol{y_n})$;
    $\left\langle \tilde{\boldsymbol{x}}_{\boldsymbol{\sigma(j)}}, \tilde{H}_{\sigma(j)} \right\rangle := F_{\sigma(j)}(t, H, \boldsymbol{x_{\sigma(j)}}, \boldsymbol{uc_{\sigma(j)}}$ or $\boldsymbol{up_{\sigma(j)}})$;
    $\boldsymbol{y_{\sigma(j)}} := G_{\sigma(j)}(t + \tilde{H}_{\sigma(j)}, \tilde{\boldsymbol{x}}_{\boldsymbol{\sigma(j)}}, \boldsymbol{uc_{\sigma(j)}})$
      or $G_{\sigma(j)}(t + \tilde{H}_{\sigma(j)}, \tilde{\boldsymbol{x}}_{\boldsymbol{\sigma(j)}})$;
  end
  $\tilde{H} := \min_{i \in D}(\tilde{H}_i)$;
  return $\left\langle [\boldsymbol{uc_1}, \ldots, \boldsymbol{uc_n}, \tilde{\boldsymbol{x}}_{\boldsymbol{1}}, \ldots, \tilde{\boldsymbol{x}}_{\boldsymbol{n}}]^T, \tilde{H} \right\rangle$;
end

Depending on the kind of external FMU being constructed, its initial state is computed by

$$Init(\boldsymbol{u_{ext}}) = [\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}, \boldsymbol{x_1}, \ldots, \boldsymbol{x_n}]^T$$
$$\text{or } Init() = [\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}, \boldsymbol{x_1}, \ldots, \boldsymbol{x_n}]^T$$

where $Init()$, to be detailed shortly, makes use of the initialization functions $Init_i$ of the internal units to get their initial states.

We now introduce the formal representation of the semantic adaptation rules, introduced at the beginning of this subsection:

- The application of the *external input rules* to the provided input is

$$In([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, \boldsymbol{u_{ext}}) = \tilde{\boldsymbol{x}}_{\boldsymbol{in}}$$

- The application of the *internal input rules* to create the internal input vector is denoted as

$$MapIn([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, h, dt) = [\tilde{\boldsymbol{u}}_{\boldsymbol{1}}, \ldots, \tilde{\boldsymbol{u}}_{\boldsymbol{n}}]^T$$

  This function is used whenever the input to any of the internal units needs to be computed. It is used in the $Ctrl$ rules (defined next) and in the output function of the external unit. In most adaptations, this function is invoked immediately before a call to the state transition function F_i of any internal unit. In line with the FMU interface, h is the communication step size that will be passed to the state transition F_i invocation, dt is the displacement of the time in unit i, relative to the external unit, and $\tilde{\boldsymbol{u}}_{\boldsymbol{i}}$ denotes the vector that will be used as external input to unit i, or ignored if the unit does not depend on the external input. Multiple calls to this function can be made: potentially one per internal state transition call.
- The application of the *control rules*, to compute the new state $\tilde{x}_i$ of each internal unit i, the step size advanced $\tilde{H}$, and the new control/output storage state $\tilde{\boldsymbol{x}}_{\boldsymbol{ctrl}}, \tilde{\boldsymbol{x}}_{\boldsymbol{out}}$ of the semantic adaptation, is

$$Ctrl(t, H, [\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, [\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}]^T) =$$
$$\left\langle \tilde{\boldsymbol{x}}_{\boldsymbol{ctrl}}, \tilde{\boldsymbol{x}}_{\boldsymbol{out}}, [\tilde{\boldsymbol{x}}_{\boldsymbol{1}}, \ldots, \tilde{\boldsymbol{x}}_{\boldsymbol{n}}]^T, \tilde{H} \right\rangle$$

This function invokes the $MapIn/MapOut$ functions before/after a state transition of an internal unit is invoked.

- The application of the *internal output rules*

$$MapOut([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, [\boldsymbol{y_1}, \ldots, \boldsymbol{y_n}]^T, h, dt) = \tilde{\boldsymbol{x}}_{\boldsymbol{out}}$$

Analogously to the $MapIn$, the invocation of this function is controlled by the $Ctrl$. Parameters h and dt denote the communication step size, and time displacement, passed as arguments to the most recently invoked state transition function F_i.

- The application of the *external output rules* to compute the external outputs, from the semantic adaptation state

$$Out([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T) = \boldsymbol{y}$$

Intuitively, the internal input/output functions serve to decouple the rate of execution of the internal units, from the rate of execution of the external FMU.

A semantic adaptation is a concrete definition of:

- Storage structure — $\boldsymbol{x_{in}}$, $\boldsymbol{x_{ctrl}}$, and $\boldsymbol{x_{out}}$;
- Initialization — $Init()$;
- External input rules — In;
- Internal input rules — $MapIn$;
- Control rules — $Ctrl$;
- Internal output rules — $MapOut$;
- External output rules — Out;

We now describe how these functions are used in the specification of an external FMU.

The generic external unit is defined exactly as a simulation unit (recall Definition 34):

$$\begin{aligned} \left\langle \boldsymbol{x}(t+\tilde{H}), \tilde{H} \right\rangle &= F(t, H, \boldsymbol{x}(t), \boldsymbol{u}_{ext}(t+H) \text{ or } \boldsymbol{u}_{ext}(t)) \\ \boldsymbol{y}(t) &= G(t, \boldsymbol{x}(t), \boldsymbol{u}_{ext}(t)) \text{ or } G(t, \boldsymbol{x}(t)) \\ \boldsymbol{x}(0) &= Init(\boldsymbol{u}_{ext}) \text{ or } Init() \end{aligned} \tag{7.5}$$

where $\boldsymbol{x} = [\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}, \boldsymbol{x_1}, \ldots, \boldsymbol{x_n}]^T$ denotes the state of the external FMU. Both an external reactive or delayed unit has the same implementation of F, described in Algorithm 10 (but note that the definition of $Ctrl$ will likely differ). The definitions of G differ for a Mealy or Moore external unit, and are detailed in Algorithm 11.

In Algorithm 11, we stress the following:

- The definitions take into account that it may not be possible to sort the internal units topologically, so the semantic adaptations support dependency cycles (recall algebraic loops in Section 3.7.3).
- Multiple calls to G can be made without changing the state of the external unit.
- If a Moore external FMU has at least one internal unit which depends on external input, then this input must be stored in the input storage $\boldsymbol{x_{in}}$ of the semantic adaptation by the In function (Line 2 of Algorithm 10), and then retrieved by the $MapIn$ function (Line 8 of Algorithm 11).

ALGORITHM 10: State transition function of the generic external FMU, defined in Equation (7.5).

Function $F(t, H, [\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}, \boldsymbol{x_1}, \ldots, \boldsymbol{x_n}]^T, \boldsymbol{u_{ext}})$
  $\tilde{\boldsymbol{x}}_{in} := In([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, \boldsymbol{u_{ext}})$;
  $\left\langle \tilde{\boldsymbol{x}}_{ctrl}, \tilde{\boldsymbol{x}}_{out}, [\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_n]^T, \tilde{H} \right\rangle := Ctrl(t, H, [\tilde{\boldsymbol{x}}_{in}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, [\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}]^T)$;
  return $\left\langle [\tilde{\boldsymbol{x}}_{in}, \tilde{\boldsymbol{x}}_{ctrl}, \tilde{\boldsymbol{x}}_{out}, \tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_n]^T, \tilde{H} \right\rangle$;
end

To make these definitions easier to understand, we provide two examples: the default reactive Mealy hierarchical unit presented in the sub-previous section, and the algebraic loop semantic adaptation that involves the obstacle and window_sa units of the power window example (loop_sa).

The default reactive Mealy hierarchical unit can be informally described as follows:

- the state $\boldsymbol{x_{ctrl}}$ of the semantic adaptation includes the previous inputs of the internal units;
- the $Init$ function is analogue to the one described in Algorithm 7;
- the In, $MapIn$, $MapOut$, and Out, are roughly identity functions;
- and the $Ctrl$ function implements the body of F, in Algorithm 9;

Formally, functions $Init$ and $Ctrl$ are defined in Algorithm 12, and:

$$
\begin{aligned}
&In([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, \boldsymbol{u_{ext}}) = \boldsymbol{u_{ext}} \\
&MapIn([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}], h, dt) = \\
&\qquad [\tilde{\boldsymbol{u}}_1, \ldots, \tilde{\boldsymbol{u}}_n]^T, \text{ with } \tilde{\boldsymbol{u}}_i = \boldsymbol{x_{in}} \\
&MapOut([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, [\boldsymbol{y_1}, \ldots, \boldsymbol{y_n}]^T, h, dt) = \\
&\qquad [\boldsymbol{y_1}, \ldots, \boldsymbol{y_n}]^T \\
&Out([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T) = \boldsymbol{x_{out}}
\end{aligned}
\tag{7.6}
$$

The second example refers to the adaptation loop_sa, which essentially performs a fixed point iteration between the obstacle and window_sa units, computing improved values for their input/outputs via successive substitution.

The external FMU, called loop_sa in Figure 7.5 is a reactive Moore unit, and has an input $\boldsymbol{u_{ext}} \in \mathbb{R}^2$ with two dimensions — displacement and speed — and one output – tau. Whenever the state transition of the external FMU is called, a successive substitution is performed between the two internal units, using the most recently found value of disp as an initial guess. Formally, let the index 1 refer to the window_sa unit, and 2 to obstacle, so that, e.g., uc_2 refers to the input to the obstacle unit. For the sake of simplicity, we assume that the system starts with all inputs/outputs being zero. Then, the functions that characterize the adaptation are shown in Equation (7.7). Note that had we not assumed that the system starts with zero inputs/outputs, the $Init$ would have to compute a fixed point iteration to find a consistent initial state. This is possible with our formalization.

The next section describes a DSL for the definition of such semantic adaptations. The examples provided in that section clarify the need for the semantic adaptation functions

ALGORITHM 11: Output functions of the generic external FMU, per kind of unit, defined in Equation (7.5).

Function $G(t, [\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}, \boldsymbol{x_1}, \ldots, \boldsymbol{x_n}]^T, \boldsymbol{u_{ext}})$
  $\tilde{\boldsymbol{x}}_{in} := In([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, \boldsymbol{u_{ext}})$;
  if σ *is defined* **then**
    for $i = 1, \ldots, n$ **do**
      $\boldsymbol{uc_i} := \boldsymbol{y_i} := \tilde{\boldsymbol{y}}_i := \boldsymbol{0}$;
    end
    for $j \in (1, \ldots, n)$ **do**
      $[\tilde{\boldsymbol{u}}_1, \ldots, \tilde{\boldsymbol{u}}_n]^T := MapIn([\tilde{\boldsymbol{x}}_{in}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, 0, 0)$;
      $\boldsymbol{uc_{\sigma(j)}} := c_{\sigma(j)}(\tilde{\boldsymbol{u}}_{\boldsymbol{\sigma(j)}}, \boldsymbol{y_1}, \ldots, \boldsymbol{y_{\sigma(j)-1}}, \boldsymbol{y_{\sigma(j)+1}}, \ldots, \boldsymbol{y_n})$;
      $\boldsymbol{y_{\sigma(j)}} := G_{\sigma(j)}(t, \boldsymbol{x_{\sigma(j)}}, \boldsymbol{uc_{\sigma(j)}})$
        or $G_{\sigma(j)}(t, \boldsymbol{x_{\sigma(j)}})$;
      $\tilde{\boldsymbol{x}}_{out} := MapOut([\tilde{\boldsymbol{x}}_{in}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, [\boldsymbol{y_1}, \ldots, \boldsymbol{y_n}]^T, 0, 0)$;
    end
  else
    $\tilde{\boldsymbol{x}}_{out} := \boldsymbol{x_{out}}$;
  end
  $\boldsymbol{y} := Out([\tilde{\boldsymbol{x}}_{in}, \boldsymbol{x_{ctrl}}, \tilde{\boldsymbol{x}}_{out}]^T)$;
  return $\boldsymbol{y}$;
end
Function $G(t, [\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}, \boldsymbol{x_1}, \ldots, \boldsymbol{x_n}]^T)$
  if σ *is defined* **then**
    for $i = 1, \ldots, n$ **do**
      $\boldsymbol{uc_i} := \boldsymbol{y_i} := \tilde{\boldsymbol{y}}_i := \boldsymbol{0}$;
    end
    for $j \in (1, \ldots, n)$ **do**
      $[\tilde{\boldsymbol{u}}_1, \ldots, \tilde{\boldsymbol{u}}_n]^T := MapIn([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, 0, 0)$;
      $\boldsymbol{uc_{\sigma(j)}} := c_{\sigma(j)}(\tilde{\boldsymbol{u}}_{\boldsymbol{\sigma(j)}}, \boldsymbol{y_1}, \ldots, \boldsymbol{y_{\sigma(j)-1}}, \boldsymbol{y_{\sigma(j)+1}}, \ldots, \boldsymbol{y_n})$;
      $\boldsymbol{y_{\sigma(j)}} := G_{\sigma(j)}(t, \boldsymbol{x_{\sigma(j)}}, \boldsymbol{uc_{\sigma(j)}})$
        or $G_{\sigma(j)}(t, \boldsymbol{x_{\sigma(j)}})$;
      $\tilde{\boldsymbol{x}}_{out} := MapOut([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, [\boldsymbol{y_1}, \ldots, \boldsymbol{y_n}]^T, 0, 0)$;
    end
  else
    $\tilde{\boldsymbol{x}}_{out} := \boldsymbol{x_{out}}$;
  end
  $\boldsymbol{y} := Out([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \tilde{\boldsymbol{x}}_{out}]^T)$;
  return $\boldsymbol{y}$;
end

defined in the current section.

$$
\begin{aligned}
&Init(\boldsymbol{u_{ext}}) = [\boldsymbol{0}, \boldsymbol{0}, \boldsymbol{0}, Init_1(\boldsymbol{0}), Init_2(\boldsymbol{0})]^T \\
&In([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, \boldsymbol{u_{ext}}) = \boldsymbol{u_{ext}} \\
&MapIn([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, h, dt) = [\boldsymbol{x_{in}}, \boldsymbol{0}]^T \\
&MapOut([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, [\boldsymbol{y_1}, \boldsymbol{y_2}]^T, h, dt) = \boldsymbol{y_1} \\
&Out([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \boldsymbol{x_{out}} \\
&\textit{Ctrl} \text{ is defined in Algorithm 13}
\end{aligned}
\tag{7.7}
$$

ALGORITHM 12: *Init* and *Ctrl* functions of the default reactive Mealy hierarchical unit.

Function $Init(\boldsymbol{u_{ext}})$
  for $i = 1, \ldots, n$ **do**
    $\boldsymbol{x_i} := \boldsymbol{up_i} := \boldsymbol{y_i} := \boldsymbol{0}$;
  end
  for $j \in (1, \ldots, n)$ **do**
    $\boldsymbol{up_{\sigma(j)}} := c_{\sigma(j)}(\boldsymbol{u_{ext}}, \boldsymbol{y_1}, \ldots, \boldsymbol{y_{\sigma(j)-1}}, \boldsymbol{y_{\sigma(j)+1}}, \ldots, \boldsymbol{y_n})$;
    $\boldsymbol{x_{\sigma(j)}} := Init_{\sigma(j)}(\boldsymbol{up_{\sigma(j)}})$ or $Init_{\sigma(j)}()$;
    $\boldsymbol{y_{\sigma(j)}} := G_{\sigma(j)}(0, \boldsymbol{x_{\sigma(j)}}, \boldsymbol{up_{\sigma(j)}})$
      or $G_{\sigma(j)}(0, \boldsymbol{x_{\sigma(j)}})$;
  end
  $\boldsymbol{x_{in}} := \boldsymbol{x_{out}} := \boldsymbol{0}$;
  $\boldsymbol{x_{ctrl}} := [\boldsymbol{up_1}, \ldots, \boldsymbol{up_n}]^T$;
  return $[\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}, \boldsymbol{x_1}, \ldots, \boldsymbol{x_n}]^T$;
end
Function $Ctrl(t, H, \left\langle \boldsymbol{x_{in}}, [\boldsymbol{up_1}, \ldots, \boldsymbol{up_n}]^T, \boldsymbol{x_{out}} \right\rangle, [\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}]^T)$
  $\boldsymbol{x_{ctrl}} := [\boldsymbol{up_1}, \ldots, \boldsymbol{up_n}]^T$;
  for $i = 1, \ldots, n$ **do**
    $\boldsymbol{uc_i} := \boldsymbol{y_i} := \boldsymbol{0}$;
  end
  for $j \in (1, \ldots, n)$ **do**
    $[\tilde{\boldsymbol{u}}_1, \ldots, \tilde{\boldsymbol{u}}_n]^T := MapIn([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, 0, 0)$;
    $\boldsymbol{uc_{\sigma(j)}} := c_{\sigma(j)}(\tilde{\boldsymbol{u}}_{\boldsymbol{\sigma(j)}}, \boldsymbol{y_1}, \ldots, \boldsymbol{y_{\sigma(j)-1}}, \boldsymbol{y_{\sigma(j)+1}}, \ldots, \boldsymbol{y_n})$;
    $\left\langle \tilde{\boldsymbol{x}}_{\boldsymbol{\sigma(j)}}, \tilde{H}_{\sigma(j)} \right\rangle := F_{\sigma(j)}(t, H, \boldsymbol{x_{\sigma(j)}}, \boldsymbol{uc_{\sigma(j)}}$ or $\boldsymbol{up_{\sigma(j)}})$;
    $\boldsymbol{y_{\sigma(j)}} := G_{\sigma(j)}(t + \tilde{H}_{\sigma(j)}, \tilde{\boldsymbol{x}}_{\boldsymbol{\sigma(j)}}, \boldsymbol{uc_{\sigma(j)}})$
      or $G_{\sigma(j)}(t + \tilde{H}_{\sigma(j)}, \tilde{\boldsymbol{x}}_{\boldsymbol{\sigma(j)}})$;
    $\tilde{\boldsymbol{x}}_{\boldsymbol{out}} := MapOut([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, [\boldsymbol{y_1}, \ldots, \boldsymbol{y_n}]^T, 0, 0)$;
  end
  $\tilde{H} := \min_{i \in D}(\tilde{H}_i)$;
  return $\left\langle [\boldsymbol{uc_1}, \ldots, \boldsymbol{uc_n}]^T, \tilde{\boldsymbol{x}}_{\boldsymbol{out}}, [\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_n]^T, \tilde{H} \right\rangle$;
end

7.5 A DSL for Semantic Adaptation

Any semantic adaptation has an essential structure: it transforms one or more FMUs into a single FMU, providing a *wrapper* implementation for the co-simulation operations.

As such, we introduce a DSL for the specification of the set of rules introduced in the previous section (which form a semantic adaptation). Since research in semantic adaptation is ongoing, the language should be expressive enough to cover future semantic adaptations. Additionally, the implementation should not violate the modularity and transparency principles.

To these ends, the DSL — named baseSA— mixes imperative concepts with convenient functions that perform common operations on simulation units. A description made in this DSL can be used to generate hierarchical units.

The language and the examples used in this paper are available for download[3].

The baseSA allows the description of the internal FMUs and their couplings (that is, the internal scenario as described in Equation (7.3)), and *how* semantic adaptation rules ($Init$, In, $MapIn$, $Ctrl$, $MapOut$, and Out), are implemented.

The remainder of this section is organised as follows. First, the baseSA DSL is introduced

[3] `https://github.com/INTO-CPS-Association/hybrid-cosim`

ALGORITHM 13: *Ctrl* function of external FMU loop_sa, illustrated in Figure 7.5.

Function $Ctrl(t, H, [\boldsymbol{x_{in}}, \boldsymbol{yp_1}, \boldsymbol{x_{out}}]^T, [\boldsymbol{x_1}, \boldsymbol{x_2}]^T)$
  $\boldsymbol{u_1} := \mathbf{0}$;
  $\boldsymbol{y_1} := \boldsymbol{yp_1}$;
  $\boldsymbol{u_2} := \boldsymbol{yp_2} := \boldsymbol{y_2} := 0$;
  $[\tilde{\boldsymbol{u}}_1, \tilde{\boldsymbol{u}}_2]^T := MapIn([\boldsymbol{x_{in}}, \boldsymbol{yp_1}, \boldsymbol{x_{out}}]^T, 0, 0)$;
  for $i \in (1, \ldots, MAX\_ITERATIONS)$ **do**
    $\boldsymbol{u_2} := c_2(\tilde{\boldsymbol{u}}_2, \boldsymbol{y_1})$;
    $\left\langle \tilde{\boldsymbol{x}}_2, \tilde{H}_2 \right\rangle := F_2(t, H, \boldsymbol{x_2}, \boldsymbol{u_2})$;
    $y_2 := G_2(t + \tilde{H}_2, \tilde{\boldsymbol{x}}_2, \boldsymbol{u_2})$;
    $\boldsymbol{u_1} := c_1(\tilde{\boldsymbol{u}}_1, \boldsymbol{y_2})$;
    $\left\langle \tilde{\boldsymbol{x}}_1, \tilde{H}_1 \right\rangle := F_1(t, H, \boldsymbol{x_1}, \boldsymbol{u_1})$;
    $\boldsymbol{y_1} := G_1(t + \tilde{H}_1, \tilde{\boldsymbol{x}}_1, \boldsymbol{u_1})$;
    if $\|\boldsymbol{y_1} - \boldsymbol{yp_1}\| \approx 0$ *and* $\|\boldsymbol{y_2} - \boldsymbol{yp_2}\| \approx 0$ **then**
      $\tilde{\boldsymbol{x}}_{out} := MapOut([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, [\boldsymbol{y_1}, \boldsymbol{y_2}]^T, 0, 0)$;
      break;
    else
      $\boldsymbol{yp_1} := \boldsymbol{y_1}$;
      $\boldsymbol{yp_2} := \boldsymbol{y_2}$;
  end
  return $\left\langle \boldsymbol{y_1}, \tilde{\boldsymbol{x}}_{out}, [\tilde{\boldsymbol{x}}_1, \tilde{\boldsymbol{x}}_2]^T, H \right\rangle$;
end

by describing the semantic adaptations of the running example and what their intended meaning is. Then, a more detailed description of the language (syntax and semantics) is provided.

7.5.1 The baseSA DSL

7.5.1.1 The window_sa adaptation

```
1  semantic adaptation reactive mealy WindowSA windowSA
2  at "./path/to/WindowSA.fmu"
3
4    for inner fmu Window window
5      at "./path/to/Window.fmu"
6      with input ports displacement (rad), speed (rad/s), reaction_force (N)
7      with output ports height (cm), reaction_torque (N.m)
8
9  output ports disp (m)  <- window.height, tau
10
11 out rules {
12   true -> {} --> {
13     windowSA.tau := -window.reaction_force;
14   };
15 }
```

Listing 7.1: The simple data adaptation window_sa in baseSA.

Listing 7.1 shows the baseSA definition of the semantic adaptation that generates the window_sa in Figure 7.5. The first few lines (Line 1 and 2 in the example) of any description, declare the name of the semantic adaptation and where the resulting external FMU will be generated.

Following that, the internal scenario is declared. The example listing (Lines 4 – 7) declares a single internal FMU and its ports.

baseSA descriptions work by *exclusion*: the user only specifies what needs to be changed, and the rest is assumed from the information provided. Hence, Listing 7.1 only needs to declare the output ports of the external FMU (`disp` and `tau`), in Line 13, and how they get their values: `disp` gets its value implicitly from the `height` port, and `tau` gets its value explicitly (via the specification of output rules).

Lines 11–15 declare the output rules. These specify how the `tau` output port of the external FMU gets its value, and this is done by assigning it the value of the `reaction_torque` output port, of the window FMU. The examples declares a single output rule, but in general multiple output rules can be declared. In general, each output rule has three parts: a condition, a *MapOutRule* part (syntactically preceded by "->"), and a *OutRule* part (syntactically preceded by "-->"). The condition decides whether the rule should be applied, and the other two parts contribute to the definition of the corresponding functions $MapOut$ and Out, respectively.

Following the exclusion principle, Listing 7.1 omits several bits of information about the external FMU, that are required for a full definition of a semantic adaptation: input ports; $Init$ function; In function; $MapIn$ function; and $Ctrl$ function;

In general, this information is assumed by applying multiple conventions, detailed in Section 7.5.3. The intended behavior is to follow the default hierarchical unit definition wherever the information is omitted (recall Equation (7.6) and Algorithm 12). For the example in Listing 7.1, the following is applicable:

- The external FMU (windowSA) has an input port for every input port of any internal FMU that has no incoming connection. This means that windowSA has three input ports, each bound to the corresponding input port of the internal FMU window.
- Each of the input ports of the internal FMU that have no incoming connections, gets its value from the corresponding external input port declared by the previous convention. The implementation of bindings is made via a storage variable. In Listing 7.1, this means that an extra input rule is created to encode the transfer of values. The input storage variables are also created.
- Any output variable bindings are realized in a manner similar to the previous convention: add an output rule and declare the necessary output variables to perform the transfer of values.
- Any expression referring ot the output of any internal FMU, in the *Out* part of an output rule, is assumed to refer to the storage variable with the most recent value of that output (output variables are created for the outputs of each internal FMU). In Listing 7.1, this means that `window.reaction_force`, in Line 13, gets replaced by a reference to an output variable.
- After applying the previous two conventions, the implicit bindings are removed.

```
1  semantic adaptation reactive mealy WindowSA windowSA
2  at "./path/to/WindowSA.fmu"
3
4    for inner fmu Window window
5      at "./path/to/Window.fmu"
6      with input ports displacement (rad), speed (rad/s), reaction_force (N)
7      with output ports height (m), reaction_torque (N.m)
8
9  input ports   reaction_force,
10           displacement,
11           speed
```

```
12
13 output ports  disp,
14         tau
15
16 param   INIT_WINDOWSA_REACTION_FORCE := 0.0,
17     INIT_WINDOWSA_DISPLACEMENT := 0.0,
18     INIT_WINDOWSA_SPEED := 0.0,
19     INIT_WINDOW_REACTION_TORQUE := 0.0,
20     INIT_WINDOW_REACTION_HEIGHT := 0.0;
21
22 control rules {
23   var H_window := do_step(window, t, H);
24   return H_window;
25 }
26
27 in var  stored_windowsa_reaction_force := INIT_WINDOWSA_REACTION_FORCE,
28     stored_windowsa_displacement := INIT_WINDOWSA_DISPLACEMENT,
29     stored_windowsa_speed := INIT_WINDOWSA_SPEED;
30
31 in rules {
32   true -> {
33     stored_windowsa_reaction_force := windowSA.reaction_force;
34     stored_windowsa_displacement := windowSA.displacement;
35     stored_windowsa_speed := windowSA.speed;
36   } --> {
37     window.reaction_force := stored_windowsa_reaction_force;
38     window.displacement := stored_windowsa_displacement;
39     window.speed := stored_windowsa_speed;
40   };
41 }
42
43 out var stored_window_reaction_torque := INIT_WINDOW_REACTION_TORQUE,
44     stored_window_height := INIT_WINDOW_REACTION_HEIGHT;
45
46 out rules {
47   true -> {
48     stored_window_reaction_torque := window.reaction_torque;
49     stored_window_height := window.height;
50   } --> {
51     windowSA.disp := stored_window_height / 100;
52   };
53   true -> { } --> {
54     windowSA.tau := -stored_window_reaction_torque;
55   };
56 }
```

Listing 7.2: The adaptation window_sa in explicit form.

Listing 7.2, on page 159, shows the same adaptation as Listing 7.1, after applying the conventions introduced above:

- All input ports and output ports of the external FMU are declared, with no implicit bindings defined.
- Input storage variables, and their initial values, are declared (`stored_windowsa_reaction_force`, and `stored_windowsa_displacement`, `stored_windowsa_speed`). These are part of the $\boldsymbol{x_{in}}$ state vector of the semantic adaptation.
- Output storage variables, and their initial values, are declared (`stored_window_reaction_torque` and `stored_window_height`), comprising part of the $\boldsymbol{x_{out}}$ state vector.
- A parameter per storage variable is added to allow the configuration of the initial value of that variable (technical detail: the parameters are mapped to FMI parameters).
- Input rules, as the one in Lines 31–41, are in general comprised of two parts: the *InRule* part, which in the example assigns values to the input storage variables; and the *MapInRule* part, which assigns the stored values to the input ports of the internal FMUs in the example. These make up the respective functions In and $MapIn$.

- The control rules make use of the special function `H_window := do_step(window, t, H)`, which automatically: uses the $MapIn$ function to compute the inputs to the internal FMU `window`, computes any extra internal input (this applies to internal interconnected units), invokes the state transition function of `window` with t and H, and invokes the $MapOut$ function to compute its outputs. `do_step` also takes into account the type (Mealy/Moore and reactive/delayed) of the internal unit invoked. The returned value is the step size taken by the unit.
- Output rules defined the functions $MapOut$ (which stores the outputs of `window` in the output storage variables), and Out (which sets the outputs of the external FMU from the output storage variables). Notice that the conversion of units between the `height` and `disp` ports is also done.

In any baseSA description, there is no need to define explicitly the initial state (computed by the $Init$ function). It is inferred from the input, control and output storage variables, plus the information about the internal units (extracted from their *xml* description file).

7.5.1.2 The loop_sa adaptation

```
1  semantic adaptation reactive moore LoopSA loop_sa
2  at "./path/to/LoopSA.fmu"
3
4    for inner fmu WindowSA window_sa
5      at "./path/to/WindowSA.fmu"
6      with input ports displacement (rad), speed (rad/s), reaction_force (N)
7      with output ports disp (m), tau (N.m)
8
9    for inner fmu Obstacle obstacle
10     at "./path/to/Obstacle.fmu"
11     with input ports disp (m)
12     with output ports reaction_force (m)
13
14   with window_sa.disp -> obstacle.disp
15   with obstacle.reaction_force -> window_sa.reaction_force
16
17 output ports tau <- window_sa.tau
18
19 param   MAXITER := 10,
20     REL_TOL := 1e-05,
21     ABS_TOL := 1e-05;
22
23 control var prev_disp := 0.0;
24 control rules {
25   var repeat := false;
26   for (var iter in 0 .. MAXITER) {
27     save_state(obstacle);
28     save_state(window_sa);
29     obstacle.disp := prev_disp;
30     do_step(obstacle,t,H);
31     do_step(window_sa,t,H);
32
33     repeat := is_close(prev_disp, window_sa.disp, REL_TOL, ABS_TOL);
34     prev_disp := window_sa.disp;
35     if (repeat) {
36       break;
37     } else {
38       rollback(obstacle);
39       rollback(window_sa);
40     }
41   }
42   return H;
43 }
```

Listing 7.3: Adaptation that generates loop_sa.

Listing 7.3 describes the adaptation defining the external FMU loop_sa in Figure 7.5. The adaptation is targeted at two internal FMUs (window_sa and obstacle) that are interconnected as specified in Lines 14–15. In general, the internal connectivity information is needed so that the generated code knows how to set the inputs to the internal FMUs. The listing does not declare input ports, therefore, according to the general conventions, the external FMU has all the input ports that that have no incoming connections (displacement and speed). A single output port is declared (tau), which gets its value from the tau output of window_sa.

Notice that the external FMU is declared as reactive Moore, and that the internal FMUs cannot be topologically sorted. Whenever this is the case, when the external output function is called, the values of the output ports returned (in the example, the value of tau) are the ones computed in the most recent state transition function.

The control block of Listing 7.3 implements Algorithm 13 with the following differences.

- As part of the semantics of the `do_step` function: the *MapInRule* and *MapOutRule* instructions (which are implicit in Listing 7.3 by convention) are executed automatically to set the inputs of the internal FMUs; and the inputs of each FMU, if unspecified by an assignment, are set according to the internal connectivity information declared in Lines 14–15.
- The convergence test (Line 33) is made only in the disp port (to simplify).
- The state manipulation of the internal FMUs is facilitated by the use of the `save_state` and rollback functions.

7.5.1.3 The rate_sa adaptation

```
1  semantic adaptation reactive moore RateSA rate_sa
2  at "./path/to/RateSA.fmu"
3
4    for inner fmu LoopSA loop_sa
5      at "./path/to/LoopSA.fmu"
6      with input ports displacement (rad), speed (rad/s)
7      with output ports tau (N.m)
8
9  input ports speed
10 output ports tau <- loop_sa.tau
11
12 param   RATE := 10;
13
14 control var previous_speed := 0;
15 control rules {
16   var micro_step := H/RATE;
17   var inner_time := t;
18
19   for (var iter in 0 .. RATE) {
20     do_step(loop_sa,inner_time,micro_step);
21     inner_time := inner_time + micro_step;
22   }
23
24   previous_speed := current_speed;
25   return H;
26 }
27
28 in var current_speed := 0;
29 in rules {
30   true -> {
31     current_speed := speed;
```

```
32      } --> {
33        loop_sa.speed := previous_speed + (current_speed - previous_speed)*(dt + h);
34      };
35    }
```

Listing 7.4: Adaptation that generates rate_sa.

The rate_sa adaptation is implemented in Listing 7.4. It is worth noticing the $MapIn$ portion of the input rules, in Line 33, which calculates the interpolation of the speed value. This function is called whenever inputs to the internal FMUs need to be provided, with $h = micro\_step$ being the communication step size asked to the internal FMU ($micro\_step$ refers to the argument used in the state transition invocation, in Line 20), and $dt = inner\_time - t$ (where $inner\_time$ is the argument used for the state transition call).

7.5.1.4 The lazy_sa adaptation

```
1  semantic adaptation reactive moore LazySA lazy_sa
2  at "./path/to/LazySA.fmu"
3
4    for inner fmu Controller controller
5    at "./path/to/Controller.fmu"
6    with input ports obj_detected, passenger_up, passenger_down, passenger_stop, driver_up, driver_down,
         driver_stop
7    with output ports up, down, stop
8
9  input ports obj_detected -> controller.obj_detected,
10       passenger_up -> controller.passenger_up,
11       passenger_down -> controller.passenger_down,
12       passenger_stop -> controller.passenger_stop,
13       driver_up -> controller.driver_up,
14       driver_down -> controller.driver_down,
15       driver_stop -> controller.driver_stop
16
17 output ports up, down, stop
18
19 param   INIT_OBJ_DETECTED := false,
20     INIT_PASSENGER_UP := false,
21     INIT_PASSENGER_DOWN := false,
22     INIT_PASSENGER_STOP := false,
23     INIT_DRIVER_UP := false,
24     INIT_DRIVER_DOWN := false,
25     INIT_DRIVER_STOP := false;
26
27 control var tn := -1.0,
28       tl := -1.0,
29       prev_obj_detected := INIT_OBJ_DETECTED,
30       prev_passenger_up := INIT_PASSENGER_UP,
31       prev_passenger_down := INIT_PASSENGER_DOWN,
32       prev_passenger_stop := INIT_PASSENGER_STOP,
33       prev_driver_up := INIT_DRIVER_UP,
34       prev_driver_down := INIT_DRIVER_DOWN,
35       prev_driver_stop := INIT_DRIVER_STOP;
36
37 control rules {
38   if (tl < 0.0){
39     tl := t;
40   }
41
42   var step_size := min(H, tn - t);
43   if (lazy_sa.obj_detected != prev_obj_detected or
44     lazy_sa.passenger_up != prev_passenger_up or
45     lazy_sa.passenger_down != prev_passenger_down or
46     lazy_sa.passenger_stop != prev_passenger_stop or
```

```
47      lazy_sa.driver_up != prev_driver_up or
48      lazy_sa.driver_down != prev_driver_down or
49      lazy_sa.driver_stop != prev_driver_stop or
50      (t+H) >= tn
51    ){
52      var step_to_be_done := (t+H-tl);
53      var step_done := do_step(controller, t, step_to_be_done);
54      tn := tl + step_done + get_next_time_step(controller);
55      step_size := tl + step_done - t;
56      tl := tl + step_done;
57    }
58
59    prev_obj_detected := lazy_sa.obj_detected;
60    prev_passenger_up := lazy_sa.passenger_up;
61    prev_passenger_down := lazy_sa.passenger_down;
62    prev_passenger_stop := lazy_sa.passenger_stop;
63    prev_driver_up := lazy_sa.driver_up;
64    prev_driver_down := lazy_sa.driver_down;
65    prev_driver_stop := lazy_sa.driver_stop;
66
67    return step_size;
68  }
```

Listing 7.5: Adaptation that generates lazy_sa.

Listing 7.5 implements adaptation lazy_sa. This adaptation assumes the default mappings for the inputs, but is declares them because they are referred to in the *Ctrl* block.

In general, every reference to an input port of the external FMU, made outside of the *In* block, is replaced with a reference to the variable that stores the most recently given value of that. For example, the expression `lazy_sa.obj_detected`, is replaced by the variable that stores that input.

The adaptation in Listing 7.5 performs two tasks: it keeps track of the previous value of each signal, and invokes the internal unit state transition function (i.e., the `do_step`) whenever there is a change; and it keeps track of the next time to execute the internal unit (assuming that no inputs change) and invokes it when such time arrives, to cater for internal timed transitions. At the same time, the output signals are always available (held constant) because of the storage output variables.

7.5.1.5 The controller_sa

```
1  semantic adaptation reactive moore ControllerSA controller_sa
2  at "./path/to/ControllerSA.fmu"
3
4    for inner fmu LazySA lazy
5    at "./path/to/LazySA.fmu"
6    with input ports obj_detected, passenger_up, passenger_down, passenger_stop, driver_up, driver_down,
         driver_stop
7    with output ports up, down, stop
8
9  input ports armature_current -> lazy.obj_detected,
10       passenger_up -> lazy.passenger_up,
11       passenger_down -> lazy.passenger_down,
12       passenger_stop -> lazy.passenger_stop,
13       driver_up -> lazy.driver_up,
14       driver_down -> lazy.driver_down,
15       driver_stop -> lazy.driver_stop
16
17 output ports  u,
18         d
19
```

```
param RTOL := 0.0001,
    ATOL := 1e-8,
    T := 5.0,
    INIT_V := 0.0;

control var c := false,
      p_v := INIT_V;
control rules {
  var step_size := H;
  var aux_obj_detected := false;
  var crossedTooFar := false;
  if ((not is_close(p_v, T, RTOL, ATOL) and p_v < T)
       and (not is_close(f_v, T, RTOL, ATOL) and f_v > T)) {
    crossedTooFar := true;
    var negative_value := p_v - T;
    var positive_value := f_v - T;
    step_size := (H * (- negative_value)) / (positive_value - negative_value);
  } else {
    if ((not is_close(p_v, T, RTOL, ATOL) and p_v < T)
         and is_close(f_v, T, RTOL, ATOL )) {
      c := true;
    }
  }

  if (not crossedTooFar){
    step_size := do_step(lazy, t, H);
  }

  if (is_close(step_size, H, RTOL, ATOL)) {
    p_v := f_v;
  }
  return step_size;
}

in var  f_v := INIT_V;
in rules {
  true -> {
    f_v := controller_sa.armature_current;
  } --> {
    lazy.obj_detected := c;
  };
}

out rules {
  lazy.up -> { } --> {controller_sa.u := 1.0; };
  not lazy.up -> { } --> {controller_sa.u := 0.0; };

  lazy.down -> { } --> {controller_sa.d := 1.0; };
  not lazy.down -> { } --> {controller_sa.d := 0.0; };

  lazy.stop -> { } --> {controller_sa.u := 0.0 ; controller_sa.d := 0.0; };
}
```

Listing 7.6: Adaptation that generates controller_sa.

The adaptation controller_sa is shown in Listing 7.6. The control rules apply *regula falsi* to locate the crossing of the armature signal into the threshold `T`.

This example shows how the conditions in the output rules can be used to select which rules are applied. Informally, in general, at the end of each external state transition, when $MapOut$ is invoked, all the conditions in the rules are evaluated. The ones that evaluate to true, are recorded as part of the x_{out} state. Afterwards, whenever Out is called, only the rules that evaluated to true contribute to the output of Out.

The power_sa adaptation was omitted due to its simplicity. It declares the external FMU as a delayed Moore and lists the output port bindings.

The above adaptations generate the FMUs for the co-simulation scenario illustrated in Figure 7.5. The master in Algorithm 6 then computes the results shown in Figure 7.6. Comparing these results with the ones in Figure 7.4, one sees that they are similar, except for the fact that the armature current has a higher peak in the co-simulation. This is because the threshold crossing adaptation was disabled, since the power FMU does not support rollback.

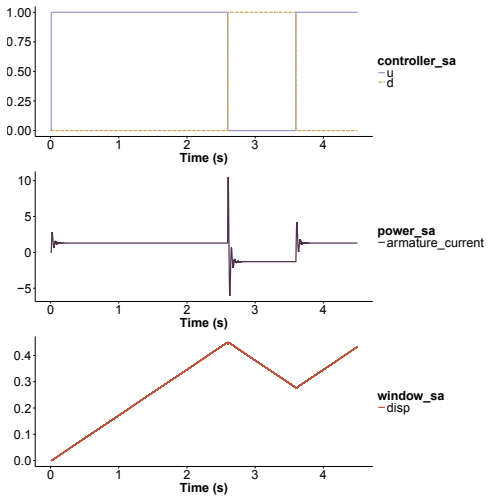


Figure 7.6: Power window co-simulation results.

In the following subsections, we describe the language (syntax and semantics) in more detail. The syntax is described using extended Backus–Naur form (EBNF) [377], and the semantics are presented informally by describing a transformation of baseSA descriptions, to the $Init$, In, $MapIn$, $Ctrl$, $MapOut$, and Out functions, introduced in Section 7.4.2.

7.5.2 Syntax

The partial syntax of baseSA is detailed in Listing 7.7. We omit the definition of the most common symbols:

- ID is an identifier;
- URL is a URL;
- PhysicalUnit denotes any physical unit;
- Expression is an expression that defines a value, e.g., comparison, addition, constant, variable reference, etc.
- Statement is a programming language statement. It includes *if*-statement, static for loop, local variable declarations, assignments, references to variables/parameters, *built-in* function calls, etc.

```
1 SemanticAdaptation = 'semantic', 'adaptation', KindInput, KindOutput, UnitName, UnitInstance,
2             'at', URL,
3             InnerUnits,
4             'input', 'ports', Port, {',', Port},
5             'output', 'ports', Port, {',', Port},
6             {ParamDeclarations}, [ControlRuleBlock], [InRulesBlock], [OutRulesBlock];
```

```
KindInput = 'reactive' | 'delayed';
KindOutput = 'moore' | 'mealy'
UnitName = ID;
UnitInstance = ID;
InnerUnits = {'for', InnerUnit} {'with', Connection};
InnerUnit = 'inner', 'fmu', UnitName, UnitInstance,
      'at', URI,
      'with', 'input', 'ports', Port, {',', Port},
      'with', 'output', 'ports', Port, {',', Port},
Port = ID, ['(', PhysicalUnit, ')'], [PortBinding];
PortBinding = ('->', ID) | ('<-', ID);
Connection = ID, '->', ID;
ParamDeclarations = 'param', SingleDeclaration, {',', SingleDeclaration};
SingleDeclaration = ID ':=' Expression;
ControlRuleBlock = {'control', VarDeclarations}, ControlRule;
VarDeclarations = 'var', SingleDeclaration, {',', SingleDeclaration};
ControlRule = 'control', 'rules', '{', {Statement}, '}';
InRulesBlock = {'in', VarDeclarations}, 'in' 'rules', '{', {DataRule}, '}';
DataRule = RuleCondition, "->", InRule, "-->", MapInRule, ';';
RuleCondition = BooleanExpression;
InRule = '{', {Statement}, '}';
MapInRule = '{', {Statement}, '}';
OutRulesBlock = RuleCondition, "->", MapOutRule, "-->", OutRule, ';';
```

Listing 7.7: The (partial) EBNF grammar of baseSA.

Table 7.1 summarizes the special functions and variables.

The full grammar definition, and an editor of baseSA descriptions, developed with Xtext [12], is available for download [4]. Figure 7.7 shows the editor interface.

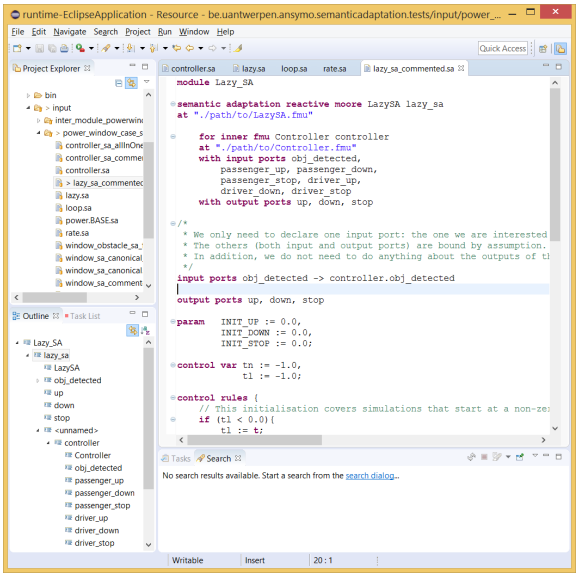


Figure 7.7: The baseSA editor.

[4] https://github.com/INTO-CPS-Association/hybrid-cosim

Table 7.1: List of *built-in* symbols and their meaning.

Symbol	Availability	Description
$t \in \mathbb{R}$	ControlRule ($Ctrl$)	Argument provided in the state transition function of the external FMU.
$H \in \mathbb{R}$	ControlRule ($Ctrl$)	Co-simulation step size passed as argument to the state transition function of the external FMU.
$do\_step(fmu, t_i, h) \in \mathbb{R}$	ControlRule ($Ctrl$)	Asks an internal FMU to perform a co-simulation step and returns the size of the computed interval.
$h \in \mathbb{R}$	MapInRule, MapOutRule ($MapIn, MapOut$)	Co-simulation step size passed as argument to the state transition function of the internal FMU.
$dt \in \mathbb{R}$	MapInRule, MapOutRule ($MapIn, MapOut$)	Let t_i denote the time given as argument to the state transition function of an internal FMU. Then $dt = t_i - t$.
$save\_state(fmu)$	ControlRule ($Ctrl$)	Stores the state of an internal FMU.
$rollback(fmu)$	ControlRule ($Ctrl$)	Rolls back an internal FMU to the last saved state.
$is\_close(x, y, rtol, atol) \in Bool$	Everywhere	Approximate equality.
$get\_next\_time\_step(fmu) \in \mathbb{R}$	Everywhere	Returns the maximum time step an internal FMU is willing to accept.
$\sin, \cos, \min, \ldots$	Everywhere	Implements the corresponding mathematical function.

7.5.3 Semantics

In this subsection, we define the semantics by describing informally how each syntactic construction in baseSA is mapped to the definition of $Init$, In, $MapIn$, $Ctrl$, $MapOut$, and Out functions, introduced in Section 7.4 (recall Figure 7.1). This is done in two stages: first we detail how any baseSA description is reduced to its explicit form; and then we describe how each baseSA description in explicit form can be mapped to the semantic functions.

7.5.3.1 Reduction to Explicit Form

Let `sa` be the name of a given baseSA description. For the sake of brevity, we make the assumption that every port has a unique name (this is not assumed by the code generator).

In order to reduce the given baseSA description to its explicit form, the following rules are applied in order, with the description resulting from the application of one rule being used in the next rule.

AddInPorts – For each input port `ip` of any internal FMU `f` that has no incoming connections, create an external input port declaration `ip -> f.ip`, if there is none already declared with the same name.

AddInParams – For each external input port declaration `ip`, create a parameter declaration `INIT_SA_IP := v` (if it does not exist already), where `v` is the default value of the parameter.

AddInVars – For each declared external input port `ip`, declare an input variable `stored_sa_ip := INIT_SA_IP` (if it does not exist) with initial value equal to the corresponding declared parameter in the previous rule.

AddInRule – Prepend a new rule to the input rules block, with a `true` condition, and: in the *InRule* part, for each declared external input port `ip`, add an assignment `stored_sa_ip := sa.ip`; in the *MapInRule* part, for each input binding declared `ip -> f.ip`, create an assignment `f.ip := ip`. If units need to be converted, the right hand side of the assignment is replaced accordingly.

RemoveInBindings – For each input binding declared `ip -> f.ip`, replace it by just `ip`. Any physical unit declaration is also removed.

AddOutPorts – If no output ports are declared, create an output port declaration `op <- f.op` per output port `op` of each internal unit `f`.

AddOutParams – Analogous to **AddInParams**: for each output port declaration `op` of each internal FMU `f`, create a parameter declaration `INIT_F_OP := v` (if such parameter does not exist), with `v` being the default value.

AddOutVars – Analogous to **AddInVars**: for each output port declaration `op` of each internal FMU `f`, create an output variable declaration `stored_f_op := INIT_F_OP`, if it does not exist already.

AddOutRule – Prepend a new output rule to the output rules block, with a `true` condition, and: in the *MapOutRule* part, add an assignment `stored_f_op := f.op`, per output port `op` of each internal unit `f`; in the *OutRule* part, for each output binding `op <- f.op` declared, add an assignment `sa.op := f.op`. If units need to be converted, the assignment is replaced accordingly.

RemoveOutBindings – Analogous to **RemoveInBindings**: for each declared output binding `op <- f.op`, remove the binding (and any unit declaration), leaving just the output declaration `op`.

CreateCtrlRules – If there is no control rules block declared, create one, and: compute the topological order σ of the internal scenario (if it cannot be computed, abort with an error); for each internal unit declaration `f`, in topological order, append `var` H_f `:= do_step(f, t, H)`; append (at the end of the block) either `return H_f` if there is only one internal FMU, or `return min(`H_1`, ...,` H_n`)`, where H_i refers to each of the local variables declared in the previous assignments.

ImplementInternalBinding – For each connection in the internal scenario `f1.op -> f2.ip`, locate the `do_step(f2, ...)` instruction in the control rules block. *Before* this instruction, if there is no assignment of the form `f2.ip := ...`, insert `f2.ip := f1.op` immediately before the instruction `do_step(f2, ...)`.

ReplacePortsRefsByVars – For every input rule, go through the *MapInRule* part and replace every reference to an external input port `ip`, by a reference to the `stored_sa_ip` input variable. In the control rules block, replace every reference to an output port `op` of an internal unit `f` by a reference to the corresponding storage variable `stored_f_op`. For each output rule, in the *OutRule* part, replace any reference to an output port `op` of an internal unit `f` by a reference to the corresponding storage variable `stored_f_op`.

Listing 7.2 is the result of applying the above rules to Listing 7.1.

7.5.3.2 Mapping to Generic Semantic Adaptation

Given a baseSA description in explicit form, we now explain how it is mapped to the formal definition of a generic external unit. In the generic external unit definition (recall Equation (7.5)), the elements that need to be defined are:

- The space of $\boldsymbol{x_{in}}$, $\boldsymbol{x_{ctrl}}$, and $\boldsymbol{x_{out}}$;
- $Init(\boldsymbol{u_{ext}})$ or $Init()$, depending on the kind of external unit;
- $In([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, \boldsymbol{u_{ext}})$;
- $MapIn([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, h, dt)$;
- $Ctrl(t, H, [\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, [\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}]^T)$;
- $MapOut([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, [\boldsymbol{y_1}, \ldots, \boldsymbol{y_n}]^T, h, dt)$;
- $Out([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T)$;

Each of the above elements are now defined.

Part of $\boldsymbol{x_{in}}$ is determined by the input variables declared: $\boldsymbol{x_{in}}$ has one dimension per declared input variable. The type of the dimension (real, boolean, etc...) corresponds to the type of the declared variable. In addition, $\boldsymbol{x_{in}}$ has one *boolean* dimension per input rule. For example, is there are three numeric variables declared, and one input rule, then $\boldsymbol{x_{in}} \in \mathbb{R}^3 \times Bool$.

Analogously to $\boldsymbol{x_{in}}$, $\boldsymbol{x_{out}}$ has one dimension per declared output variable, and an additional *boolean* dimension per declared output rule.

The control storage vector $\boldsymbol{x_{ctrl}}$ has one dimension per declared control variable. Additionally, if the semantic adaptation is a reactive one and the initial baseSA description (not the explicit one) does not include any control rules, the $\boldsymbol{x_{ctrl}}$ has one dimension per internal delayed unit.

The external input function

$$In([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, \boldsymbol{u_{ext}}) = \tilde{\boldsymbol{x}}_{\boldsymbol{in}}$$

is defined to perform the the following steps in order:

1. Evaluate all conditions of the input rules in the order that they are declared, and for each condition, mark the corresponding location of $\tilde{\boldsymbol{x}}_{\boldsymbol{in}}$ with the outcome (true or false).
2. For the input rules whose conditions evaluated to true in the previous step, execute the *InRule* part, in the order that the rules are declared (this computes the remainder of $\tilde{\boldsymbol{x}}_{\boldsymbol{in}}$).

Function

$$MapIn([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, h, dt) = [\tilde{\boldsymbol{u}}_{\boldsymbol{1}}, \ldots, \tilde{\boldsymbol{u}}_{\boldsymbol{n}}]^T$$

executes the *MapInRule* part of the input rules whose condition evaluated to true (this information is stored in $\boldsymbol{x_{in}}$) in order of their declaration. The executed input port assignments form $[\tilde{\boldsymbol{u}}_{\boldsymbol{1}}, \ldots, \tilde{\boldsymbol{u}}_{\boldsymbol{n}}]^T$.

Function

$$MapOut([\boldsymbol{x_{in}}, \boldsymbol{x_{ctrl}}, \boldsymbol{x_{out}}]^T, [\boldsymbol{y_1}, \ldots, \boldsymbol{y_n}]^T, h, dt) = \tilde{\boldsymbol{x}}_{\boldsymbol{out}}$$

is analogous to In. It evaluates all the conditions of the output rules in the order that they are declared, and for each of those conditions, marks the appropriate location of $\tilde{\boldsymbol{x}}_{\boldsymbol{out}}$ with the outcome of the condition evaluation. Then it computes the remaining portion of $\tilde{\boldsymbol{x}}_{\boldsymbol{out}}$ by executing the *MapOutRule* part of each of the output rules whose conditions evaluated to true.

Function

$$Out([\boldsymbol{x}_{\boldsymbol{in}}, \boldsymbol{x}_{\boldsymbol{ctrl}}, \boldsymbol{x}_{\boldsymbol{out}}]^T) = \boldsymbol{y}$$

is analogous to $MapIn$. It executes the *OutRule* part of the output rules whose condition evaluated to true (in the order in which they are declared) to compute the output vector $\boldsymbol{y}$.

The role of the initialization function (derived automatically from the baseSA description) is to find a consistent initial state, defining the initial values of the storage vectors $\boldsymbol{x}_{\boldsymbol{in}}$, $\boldsymbol{x}_{\boldsymbol{out}}$, and $\boldsymbol{x}_{\boldsymbol{ctrl}}$. If the semantic adaptation is declared as reactive, then $Init$ requires the initial input, according to Definition 34.

First, the parts of $\boldsymbol{x}_{\boldsymbol{in}}$, $\boldsymbol{x}_{\boldsymbol{ctrl}}$, and $\boldsymbol{x}_{\boldsymbol{out}}$ that correspond to the declared input/control/output variables are initialized according to the initial value that is declared for them.

If it exists, the part of $\boldsymbol{x}_{\boldsymbol{ctrl}}$ that corresponds to the previous inputs to the internal units is initialized by compting the initial input to all the internal units in the topological order (such order exists by assumption). This is similar to Algorithm 7, except that the functions In, $MapIn$, and $MapOut$, are invoked to adapt any external input to the internal units, and initialize the condition flags.

Function

$$Ctrl(t, H, [\boldsymbol{x}_{\boldsymbol{in}}, \boldsymbol{x}_{\boldsymbol{ctrl}}, \boldsymbol{x}_{\boldsymbol{out}}]^T, [\boldsymbol{x}_{\boldsymbol{1}}, \ldots, \boldsymbol{x}_{\boldsymbol{n}}]^T) = \left\langle \tilde{\boldsymbol{x}}_{\boldsymbol{ctrl}}, \tilde{\boldsymbol{x}}_{\boldsymbol{out}}, [\tilde{\boldsymbol{x}}_{\boldsymbol{1}}, \ldots, \tilde{\boldsymbol{x}}_{\boldsymbol{n}}]^T, \tilde{H} \right\rangle$$

runs the instructions declared in the control rules block, in the order that they are declared. The assignments performed to control variables make up part of the output vector $\tilde{\boldsymbol{x}}_{\boldsymbol{ctrl}}$. The executed assignments to the input ports of each internal FMU i, up to the instruction $do\_step(i, t_i, h_i)$, make up part of the unit input vector $\boldsymbol{u}_{\boldsymbol{i}}$.

Any variable reference in the control rules block refers to the most recently given value of that variable.

Each instruction $do\_step(i, t_i, h_i)$ maps to the following steps, performed in $Ctrl$:

- Invoke $MapIn$ function to compute the external input of unit i:

$$[\ldots, \tilde{\boldsymbol{u}}_{\boldsymbol{i}}, \ldots]^T := MapIn([\boldsymbol{x}_{\boldsymbol{in}}, \tilde{\boldsymbol{x}}_{\boldsymbol{ctrl}}, \boldsymbol{x}_{\boldsymbol{out}}]^T, h_i, t_i - t) \tag{7.8}$$

  Note that $\tilde{\boldsymbol{x}}_{\boldsymbol{ctrl}}$ represents the control state vector that was affected by the assignments made since the beginning of the execution of the $Ctrl$ function. $\boldsymbol{x}_{\boldsymbol{in}}$ and $\boldsymbol{x}_{\boldsymbol{out}}$ represent the (unchanged) vector provided as input to $Ctrl$.
- Merge the input vector $\boldsymbol{u}_{\boldsymbol{i}}$ computed by previous assignments with $\tilde{\boldsymbol{u}}_{\boldsymbol{i}}$ to form the unit input $\boldsymbol{uc}_{\boldsymbol{i}}$;

- Invoke the state transition function of the unit:

$$\left\langle \tilde{\boldsymbol{x}}_{\boldsymbol{i}}, \tilde{H}_i \right\rangle := F_i(t, H, \boldsymbol{x}_{\boldsymbol{i}}, \boldsymbol{uc}_{\boldsymbol{i}} \text{ or } \boldsymbol{up}_{\boldsymbol{i}}) \tag{7.9}$$

- Get the output of the unit:

$$\boldsymbol{y}_{\boldsymbol{i}} := G_i(t + \tilde{H}_i, \tilde{\boldsymbol{x}}_{\boldsymbol{i}}, \boldsymbol{uc}_{\boldsymbol{i}}) \text{ or } G_{\sigma(j)}(t + \tilde{H}_i, \tilde{\boldsymbol{x}}_{\boldsymbol{i}}) \tag{7.10}$$

- Invoke the $MapOut$ function to compute an updated output storage vector:

$$\begin{aligned} &\tilde{\boldsymbol{x}}_{\boldsymbol{out}} := \\ &\quad MapOut([\boldsymbol{x}_{\boldsymbol{in}}, \tilde{\boldsymbol{x}}_{\boldsymbol{ctrl}}, \boldsymbol{x}_{\boldsymbol{out}}]^T, [\boldsymbol{y}_{\mathbf{1}}, \ldots, \boldsymbol{y}_{\boldsymbol{n}}]^T, h_i, t_i - t) \end{aligned} \tag{7.11}$$

Finally, upon returning, if the external FMU is a reactive unit, and the initial baseSA description does not declare a control rules block, $Ctrl$ stores the most recent inputs provided to each delayed internal units in the $\tilde{\boldsymbol{x}}_{\boldsymbol{ctrl}}$ vector, to be used as delayed inputs in a subsequent external state transition call. This instruction is similar to Line 29 of Algorithm 12.

7.6 Evaluation

In this section, we judge how well our approach answers the research question posed in this work.

The requirements set by the research question are:

Productivity – Does the language have impact in the productivity of its users?
Expressivity – Is the language expressive enough to cover *current* and *future* needs?
Modularity – Does the internal FMUs need to be changed?
Transparency – Does the external FMU behave exactly as an FMU?

7.6.1 Productivity

In general, DSLs have the potential to boost its users' productivity [205, 208]. For baseSA, we describe an early experiment to assess the productivity.

7.6.1.1 Goals

Productivity is measured by comparing the time it takes for a *trained user* to: (1) create an external FMU using our DSL; and (2) code the same external FMU.

As a surrogate measure, we compare the approximate number of lines of code (LOC) required for a semantic adaptation coded by hand, with the LOC of the corresponding semantic adaptation expressed in baseSA.

7.6.1.2 Experimental Setup

As part of the development of the code generator, all semantic adaptations identified in Figure 7.5, except the rate_sa, were coded by hand and the effort taken was recorded.

7.6.1.3 Results

Table 7.2 shows the adaptation, the approximated lines of code (LOC), and the effort in coding the semantic adaptations in C.

Table 7.2: Effort in hand-coding hierarchical semantic adaptations.

Semantic Adaptation	LOC	Effort (man-hour)
lazy_sa	700	9
controller_sa	750	24
power_sa	680	16
window_sa	690	8
loop_sa	690	16
Total	3510	73

As Table 7.2 shows, even though the semantic adaptations differ in complexity, they have a similar number of LOC. This is evidence that there is a large portion of code dedicated to common FMI-related management tasks. With baseSA, the user does not have to code:

- Memory management – The inputs, outputs, and local variables, of the external FMU are stored in dynamically allocated memory.
- Variable de-referencing – To set/get values to/from an internal FMU, a list of value references (integers which identify a variable) has to be provided. Any mistake here may cause the internal FMU to give wrong results, but not necessarily crash, which makes it hard to debug.
- State management – The external FMU has to support rollback, and for that, the state variables must be properly serialized and de-serialized. In the case study, each semantic adaptation requires approximately 140 LOC to implement the set/get state.
- Consistent inputs management – The external FMU which is reactive and has internal delayed units, has to keep track of the previous inputs to these.

7.6.1.4 Threats to Validity

LOC is only a surrogate measure for the productivity of a DSL, albeit a common one [50], and depends on the programmer. However, the tasks described in the above list are handled automatically by the code generator of baseSA.

The values provided in Table 7.2 lack external validation. We intend to perform a second round of experiments, where we will ask a participant to code a semantic adaptation, then train him/her, and measure the effort it takes to code the same adaptation, in baseSA.

7.6.2 Expressivity

The baseSA DSL is imperative in the sense that it describes *how* the semantic adaptations are performed. However, it forces a structure in the definition of the semantic adaptations, aided by the distinction between data (input/output rules) and control adaptations. We argue that this structure does not restrict the expressiveness of the semantic adaptations.

To provide evidence for this, we describe how the adaptations used in the case study are representative of the semantic adaptations and coupling algorithms surveyed in [157].

Extrapolation/interpolation schemes These techniques, used in [39, 64, 69, 70, 71, 116, 339], are similar to the rate_sa.

Jacobi-based master This master algorithm, used in [35, 73, 120, 132, 136, 165, 214, 375], is similar to the Gauss-seidel coupling except that it assumes that all units are delayed. A way to implement it as a semantic adaptation is to define a control rule that sets explicitly the inputs to the internal FMUs, and then invokes the `do_step` function on them.

Algebraic constraint couplings This coupling technique, reported in [23, 166, 329, 332], can be implemented by a fixed point iteration (recall adaptation loop_sa) and extra algebraic computations on the units inputs and outputs.

Semi-implicit coupling These techniques, presented in [324, 326, 327, 328, 329], are similar to the ones above, except they perform two iterations only.

Error control Richard extrapolation [24, 26, 136] can be implemented by creating a semantic adaptation which runs a whole scenario at twice the rate of the original one; Multi-order input extrapolation [73, 75] amounts to implementing two approximation schemes (see item above) and run in parallel; Embedded method [184] requires that a semantic adaptation is implemented to perform a discretized numerical integration of some of the signals in the internal scenario; Energy based [318] techniques can be implementing by coding semantic adaptations which monitor for energy dissipativity in some of the signals in the internal FMUs.

We do restrict the expressiveness of the language, with the intent of guaranteing that it terminates:

- No function definitions are allowed;
- No recursive definitions of semantic adaptations are allowed;
- For loops must have a static range.

These restrictions make expressing some of the above techniques more cumbersome, but not impossible.

7.6.3 Modularity

The simulation unit specification, introduced in Definition 34 was shown to be a valid abstraction of an implementation of an FMU in Section 7.2. Furthermore, it is clear that changing the implementation of any of the functions $Init, G, F$ implies a change in the FMU implementation. In Section 7.5.3, these functions are invoked as part of the implementation of each semantic adaptation, but never changed, thus showing that the corresponding FMU implementations are not affected by the implementation of the language.

7.6.4 Transparency

Section 7.4 describes how a generic semantic adaptation forms a simulation unit that obeys Definition 34 (see Equation (7.5)). Furthermore, Section 7.5.3 describes how a baseSA is implemented by "filling in" the semantic adaptation functions, that are used in Section 7.4. The semantics does not require the hierarchical unit definition, in Equation (7.5), to be changed. Therefore, our approach does not violate transparency.

7.7 Discussion and Future Work

This section discusses some of the characteristics and limitations of our contribution, and research opportunities for the future.

Automatic Semantic Adaptation Identification. Throughout this work, we assumed that the user knows that an adaptation is required in order to make the co-simulation possible. An interesting research direction is to explore what means can be employed in trying to identify the need for specific semantic adaptations.

Runtime Performance. Despite not being our primary goal, the performance of the generated FMU should be similar to a custom coded one. To this end, the code generator under development performs most tasks at compile time. However, we have not carried out any experiments to measure the performance of the generated code.

A research direction is to explore how to merge multiple adaptations, to avoid generating the intermediate hierarchical FMUs. For example, in Figure 7.5, adaptations loop_sa and rate_sa could be merged into one single adaptation, provided that the user has no intention of using loop_sa for other purposes. However, while it is clear what the result should be for this example, in general this is non-trivial task: *When can two arbitrary adaptations be merged?*

Solving this problem brings a performance benefit, but also provides new insights into the nature of adaptations. In addition, one can ask: *If two semantic adaptations can be combined, are they commutative?* This question is important because it allows us to optimize: if there is a semantic adaptations which will cause rollbacks, we want it to be the first to execute, to avoid wasting computation. An example of this is the controller_sa, which performs the crossing location before the *lazy_sa* gets the opportunity to run.

Trying to answer the above questions will inevitably lead to another question: *what is the right level of abstraction to analyse the combination of semantic adaptations?* This question is related to the next discussion topic.

Usability and Productivity. As part of trying to find out what the right level of abstraction to analyze semantic adaptations is, we are developing a new language that allows for a more declarative description of semantic adaptations. This language, as opposed to baseSA, allows for a much more concise description of the most common semantic adaptations by just enumerating *what* semantic adaptations should be used to form the external FMU.

The descriptions made in this language compile to baseSA, whose role is to provide a solid foundation.

The main benefits of using this language are:

1. The user does not to know how semantic adaptations are implemented.
2. It is minimal, meaning that it enables the user to specify common semantic adaptations (e.g., multi-rate, successive substitution) as concisely as describing them in natural language;
3. It further restricts the user into using well known semantic adaptations, which prevents mistakes.

4. It may provide insight into the research questions identified in the previous subsection.

```
1 import PowerWindowModel
2
3 semantic adaptation reactive moore RateLoopSA rate_loop
4 at "./path/to/RateLoopSA.fmu"
5 for fmu WindowSA windowSA, Obstacle obstacle
6 successive substitution starts at height with absolute tolerance = 1e-8 and relative tolerance = 0.0001
7 multiply rate 10 times with first order interpolation
```

Listing 7.8: Example description in higher level semantic adaptation DSL.

Listing 7.8 shows an example of what such DSL looks like. The syntax reuses part of the syntax of baseSA. The description of the FMUs can be done in a separate module, which is then imported (Line 1). For this example, the FMUs are described as in Lines 4–15 of Listing 7.3. After the preliminaries, the description of each semantic adaptation occupies one line (Line 6 for loop_sa, and Line 7 for rate_sa). In this language, adaptations are applied in order, meaning that the outer most adaptation is the multi-rate one.

Each adaptation has some degree of configuration. For example, the multi-rate is configurable with an input approximation adaptation. This highlights another interesting research direction, related to the combination of semantic adaptations: *how and when can semantic adaptations interface with each other?* In this example, it is clear that any input approximation adaptations can complement a multi-rate adaptation, but what are the essential characteristics of input approximation and multi-rate adaptations, that make them so compatible? The same question applies to output approximation adaptations (the family of Hold adaptations) and the lazy related ones. A possible direction to explore is to look at the object oriented world, and study how can semantic adaptations define interfaces and specialization, so that their interaction is well defined.

Discrete Event FMU Implementation. The current version of the FMI standard (version 2.0) lacks essential features to enable accurate hybrid co-simulation (see, e.g., [66, 67, 88, 98, 122]).

Until new extensions are made, there are many different ways in which a cyber system (e.g., a state chart) can be simulated in an FMU [98, 100, 122, 288, 303, 353]. At least one of the implementations the authors used before (the Stategraph [288]), already includes semantic adaptations, to facilitate its integration with the FMI.

Our work shows that, when implementing an FMU that simulates a cyber system, it is best to leave as many semantic adaptations as possible out. The more adaptations an FMU already contains, the harder it is to adapt it to other contexts.

7.8 Related Work

Outside the context of FMI, the problem of composing and adapting operational semantics of multiple languages is discussed in [60, 61, 103, 105, 222, 251, 267, 371] and references therein.

Within the context of FMI, we can divide the related works in two categories: (A) those whose prime purpose is to describe co-simulation scenarios; and (B) those that target the

description of master algorithms. Both these categories do not target primarily the description of semantic adaptations, but can potentially be extended to include simple descriptions. Due to our pure hierarchical co-simulation approach, our contribution complements any of these works.

Under Category (A), we highlight [358], [226], [136], and [77]. These works introduce a language for the description of a co-simulation scenario, with the purpose of running a co-simulation. The work in [136, 226] assumes that a generic master algorithm is used, whereas [77, 358] aim at generating an master that is specific to the scenario described. Our DSL allows for the description of a co-simulation scenario, and a specific master algorithm can be generated from that description.

DACCOSIM [136] follows a related approach with respect to hierarchical co-simulation, allowing the scenario to grouped by computational nodes. In contrast to our work, this hierarchy is computational and not functional. Moreover, it is not transparent, as the distinction is made between local (internal to computational nodes) and global master algorithms. Nevertheless, each FMU is wrapped with code that performs error control, highlighting the need for semantic adaptation.

In Category (B), we highlight [77], [143], and [27]. The work in [77] allows the description of master algorithms using the Business Process Modelling Notation. We argue that the visual notation for the description of an master algorithm works well for simple cases, with two units. However, when multiple semantic adaptations become necessary, or the number of simulation units increases, the visual notation rapidly becomes cluttered. The work does not describe any intention of using the notation to describe semantic adaptations, but the notation has an extension mechanism that can in principle be used to describe simple semantic adaptations.

The most related to our own is [27]. It introduces an object oriented framework for co-simulation that allows for both the development of FMUs, as well as for master algorithms, in C++. Class specialization is used extensively to maximize reuse, sharing some of the benefits with our contribution. The main difference to our work is the level of abstraction and the intention to use semantic adaptations. While their work is capable of expressing semantic adaptations, our work is targeted towards that purpose. One can position their work as helping develop FMUs for simulators that need to support the FMI Standard, and our work can be used to adapt already existing FMUs. Furthermore, the description of a complex adaptations such as rate_sa is more compact in our DSL.

7.9 Concluding Remarks

This chapter addressed the problem of describing the most common semantic adaptations on multiple types of black box simulation units in a productive manner while avoiding the modification of the units (modularity) and tools for co-simulation (transparency).

To make this possible, we propose a DSL, available for download[5], that is both expressive (due to its imperative nature) but also productive (due to its conventions and high level constructs). Each description refers to a group of interconnected FMUs and dictates how those FMUs interact with the environment.

[5] https://github.com/INTO-CPS-Association/hybrid-cosim

The essential mechanism that enables the semantic adaptations is the concept of hierarchical co-simulation, formalized in this work. The meaning of each adaptation is given by mapping it onto hierarchical co-simulation units, which in turn is mapped to units and FMUs, as illustrated in Figure 7.1.

The main distinguishing factor from the related work, is our focus in semantic adaptations for FMI based co-simulation, which imposes the modularity and transparency requirements.

Limitations and Assumptions. Regarding the implementation of the FMU, we assumed that we know how its inputs are handled, and how the outputs depend on the inputs. Moreover, we assumed that FMUs can reject the step size and inform us of the largest step size they can take. We also did not consider the initialization problem. It can be treated as a separate problem, so the results presented here do not lose their generality. We assumed that FMUs rollback, because that is crucial for many semantic adaptations. In practice, many FMUs do not support this feature, as it is optional in the FMI standard.

This work opens up new opportunities for research into semantic adaptations, for example, how to find higher levels of abstraction to describe semantic adaptations, and explore how different semantic adaptations can interface and complement each other. We intend to explore these in the future.

Chapter 8

Hint-Based Configuration of Co-simulations

Disclaimer The content in this chapter is adapted from:

- Gomes, Cláudio, Bentley James Oakes, Mehrdad Moradi, Alejandro Torres Gamiz, Juan Carlos Mendo, Stefan Dutre, Joachim Denil, and Hans Vangheluwe. 2019. "HintCO - Hint-Based Configuration of Co-Simulations." In International Conference on Simulation and Modeling Methodologies, Technologies and Applications, *best student paper award*. Prague, Czech Republic.

As discussed in Chapters 2 to 4, the black-box nature of co-simulation amplifies the difficulties in ensuring that the results can be trusted. This chapter focuses on exploring the problem of configuring a co-simulation in a way that preserves the qualitative properties of the original system. This is recognized as one of the challenges faced in industry, a fact that we confirmed with our industrial partners.

8.1 Introduction

Co-simulation is typically used as part of: an optimization loop (e.g., design space exploration), physical system integration analysis (e.g., hardware-in-the-loop), and/or impact analysis of sub-model refinements. In any of these use cases the coupled models being simulated are under constant change. Therefore, the person interested in the results of the analysis, henceforth denoted as the user, may be unable to properly configure each individual co-simulation.

Moreover, when a co-simulation result is incorrect (see Section 8.3 for a more rigorous definition), there can be multiple causes [23, 24, 159, 325]:

- Sub-models are incorrect;
- FMUs use the incorrect simulation algorithm;
- The master algorithm is incorrect.

This means that the user has to be familiar with a wide range of domains in order to correctly configure the co-simulation.

For example, an FMU can be labeled as a software controller allowing the framework to generate master algorithms that: 1) respect the execution rate of the FMU, and 2) ensure the causality of software execution in digital platforms. Further examples are given in Section 8.2.

The survey described in Chapter 4 corroborates the fact that users do not always know how to configure the co-simulation.

In this chapter, we propose a way to tackle the challenge of configuring co-simulations, formalized in Section 8.3. Motivated by discussions with our industrial partners, we noticed that, while users may not know how to configure the co-simulation, they have intuition about the behaviour of the system and when the simulation result is not correct.

Hence, we propose a language to describe hints, i.e., properties about the co-simulation scenario and coupled model, and a framework, called *HintCO*, that uses those hints to propose co-simulation master algorithms that are good candidates to produce correct results.

We make use of the results introduced in Chapter 7, and the good practices identified in Chapter 3.

Figure 8.1 lays out the HintCO framework. There are three main components:

a) HintCO Hint Language which allows the user to specify their expectations for the results of a correct co-simulation (Section 8.4). Common hints are provided in a built-in library so the user may easily choose and adapt them to a specific co-simulation.

b) Generation of Candidate Master Algorithms which is the method for mapping a given set of hints to sets of master algorithms (Sections 8.5.1 and 8.5.2). In short, the hints provided induce a *search space* of possible master algorithms, and a ranking of the most important features for a good master algorithm.

c) Execution of the Master Algorithms where each master algorithm produced by the search is executed (Section 8.5.3). The results are presented to the user for inspection.

In Section 8.6, we discuss other approaches that complement our own, and Section 8.7 summarizes of our research and the steps to extend our framework further.

8.2 Industrial Example

This section describes the added value of co-simulation for our industrial partners. Then, we introduce the case study made available by Boeing and illustrate the challenge of finding the correct configuration for the co-simulation. Finally, we argue that to know which configuration is likely to be the best, we need domain knowledge.

8.2.1 Value of Co-simulation for Boeing

Boeing's vision on the Digital Twin era of aviation involves the integration of models coming from different physical domains, software environments and numerical characteristics into a single virtualized aircraft [51]. This vision requires the creation of unified modeling

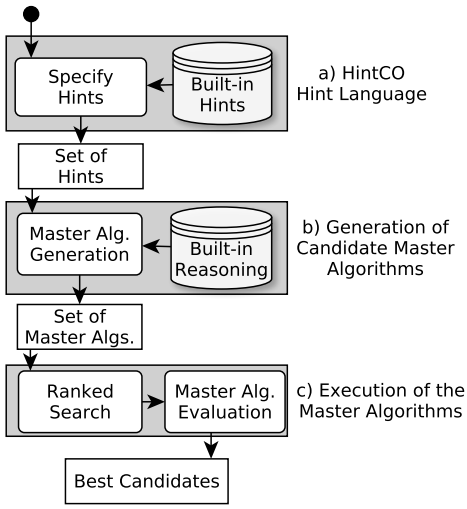


Figure 8.1: HintCO framework overview.

environments, where engineers can seamlessly evaluate the impact of a local modifications in the global system. However, these systems are comprised of many heterogeneous models, which cannot be integrated seamlessly in a single monolithic simulation. As such, Boeing regards co-simulation as one of the key technologies to enable the Digital Twin vision.

8.2.2 Boeing's Case Study

The case study presented was developed by Boeing. It constitutes a representative generic Flight Controls System, in the form of a co-simulation scenario. The FMUs are black boxes, having only the description of the input/output variables, and parameters. Moreover, no source code was made available, thereby protecting Boeing's Intellectual Property (IP), and the parameters given do not represent accurate values.

The IP-protected case study was shared with the University of Antwerp for co-simulation optimization, and the circumstances represent a faithful reproduction of Supplier–OEM relationships, where IP management tends to be an issue.

Case Study Consider a control system, represented as a co-simulation scenario in Figure 8.2. The `Controller` FMU represents a software controller, the `Plant` and `Load` FMUs represent the physical subsystems. The `Environment` FMU produces a constant signal for `psu`, and a step signal for `ref`.

We do not have access to the correct behavior of the co-simulation scenario described in Figure 8.2. However, the `Load` FMU has been designed and tested against abstractions of the `Plant` and `Load` FMUs, hence we can assume that its behavior should not be fundamentally different in the co-simulation. Moreover, the `Plant` and `Load` FMUs are produced by specialized teams, which know how the behavior of the corresponding subsystems should look like. Therefore, we make the following assumptions.

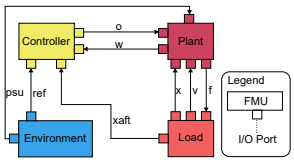


Figure 8.2: Case study co-simulation scenario.

Assumption 12. *The* `Controller` *FMU is a software controller designed for a sample rate of* 1×10^6 Hz. *The* `Plant` *and* `Load` *FMUs model physical subsystems connected by a power connection, where* `v` *represents the effort, and* `f` *the flow.*

Assumption 12 represents the domain specific knowledge that users of co-simulation use to judge the correctness of the results. For example, the movement of the `Plant` and `Load` subsystems should be smooth.

Assumption 13. *The FMUs do not support rollback, never reject a step size, and do not have I/O feed-through information.*

Assumption 13 reflects the fact that the FMU providers have implemented only the mandatory part of the FMI Standard.

Assumption 14. *The* `Controller`*,* `Plant`*, and* `Load` *FMUs are correct in the sense that, if they are provided with valid inputs, they will produce valid outputs with respect to their intended function. That is, the FMUs are correctly built.*

Assumption 14 means that, if the co-simulation results are not accurate, it is only because the co-simulation is not correctly configured.

The precise definition of co-simulation configuration is found in Section 8.3. In brief, the configuration includes the order in which the outputs are propagated to the inputs, the order of execution of each FMU, and the size of communication step. Even for small systems, the number of possible configurations can be infinite.

8.2.3 Analysis

The paragraphs that follow discuss why most co-simulation algorithms will fail to accurately reproduce the behavior of the co-simulation scenario in Figure 8.2. This is illustrated by showing two representative co-simulation algorithms that produce incorrect results with respect to assumption 12, which details information about FMUs and signals.

Experiment 1. Taking the hint that the `Controller` needs to sample the system every 1×10^{-6} s, the first co-simulation algorithm we apply is the fixed-step-Jacobi. This algorithm keeps the FMUs in sync by propagating outputs to inputs before asking each FMU to compute the next interval [35].

Figure 8.3 shows the output computed by this algorithm for the `Load` FMU, which is examined because it is the most sensitive to the master algorithm. As clearly evident in the figure, the trace produced is not smooth.

Experiment 2. A user's intuition may be that the `Load` and `Plant` FMUs need to communicate at a higher rate than the sample of the `Controller` FMU due to assumption 12. Hence, we apply a multi-rate co-simulation algorithm such as the one described in [358], to produce the results shown in Figure 8.4. In this trial, we chose the communication rate of

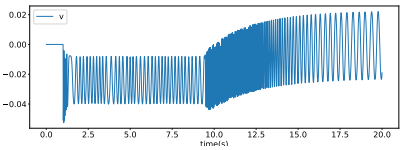


Figure 8.3: Output of `Load` FMU in experiment 1. Step size is 1×10^{-6} s.

the `Load` and `Plant` FMUs to be ten times higher. However, the result is still not smooth.

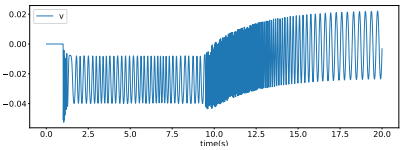


Figure 8.4: Output of `Load` FMU in experiment 2. Communication rate between `Plant` and `Load` is 1×10^{7} Hz.

The results of experiments 1 and 2 suggest that an adaptive step size co-simulation algorithm (e.g., [24, 48, 75, 317]) will fail: the step size of 1×10^{-7} s is already the minimum that can be used before the run-time execution time becomes intolerable by our industrial partners. For reference, the result in Figure 8.4 takes on average 32 minutes to compute on a Core i7 3.5GHz laptop.

To address the run-time execution issue, we turn to corrective co-simulation approaches: either a global error correction technique is used, or the input approximation of each FMU can be improved, so that less error is introduced. However, the global error correction technique cannot be applied, because the co-simulation scenario includes FMUs whose output is discontinuous (`Scenario`, and `Controller`). This thus violates the continuity assumptions that both these techniques make.

Experiment 3. To improve the input approximations on each FMI, a Gauss-Seidel co-simulation algorithm [35] can be employed to determine whether interpolations can be used instead of extrapolations on some of the FMUs. This algorithm executes each FMU in order, using the most recently computed outputs to feed the FMUs that still need to be executed.

For example, at the beginning of each co-simulation step, the `Load` FMU is given the output produced by the `Plant` and asked to compute the next interval. Then, the output `xaft` is propagated to the `Controller`, which is then asked to computed the next interval, and so on.

Figure 8.5 shows the results of this experiment. As can be seen, the trajectory is preferable to the other experiments, but is still not smooth enough for the system to be considered properly configured.

The Gauss-Seidel algorithm is more difficult to configure because it assumes that some FMUs can be executed with inputs "from a future simulated time". This is not the case for the `Controller`, because a software controller cannot predict the state of the `Load` or `Plant` FMU 1×10^{-6} s ahead. Similarly, the `Plant` can only react to a change in

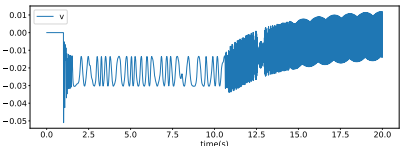


Figure 8.5: Output of `Load` FMU in experiment 3. Step size is 1×10^{-6} s, the signals `x` and `v` are extrapolated, and `f` is interpolated.

the output of the software controller after this change has occurred (this is formalized in Section 8.3). For the remaining FMUs used in the case study, causality is not an issue, because we are anyhow relaxing the higher frequencies at which they affect each other in reality [215]. Moreover, higher-order input approximation techniques can only be applied to signals that are continuous (i.e., signals `x`,`v`, and `f`).

These restrictions imply that most state-of-the-art algorithms relying on correction of signals (e.g., [40, 43, 161]) cannot be applied without some form of configuration.

8.3 Problem Formulation

In this section, we detail our research problem. In brief, we search through the configuration parameters of the co-simulation master algorithm in order to generate good co-simulation candidates. Preferable candidates more closely match the user's intuition of the properties of the system, as expressed through the hints.

8.3.1 Co-simulation Formalization

In this section, we define some core concepts that allows us to formalize the notion of model behavior approximations, when there is no reference solution.

A *dynamical model* has the purpose of approximating the relevant behavior of the original system with respect to some properties of interest, denoted by P. We assume every dynamical model has a *behavior trace*, which is the set of trajectories followed by the state (and outputs) of a dynamical model. We refer to the time variable $t \in \mathbb{R}$ as *simulated time*—or simply *time*, when no ambiguity exists.

When the behavior trace of dynamical model M satisfies property $p \in P$, we write $M \vDash p$.

Definition 35. A dynamic model is *valid* when its behavior satisfies the same properties as the original system.

Remark 12. The satisfaction relation $\vDash$ used in Definition 35 provides a binary result. In practice, a model M can partially satisfy a property p. For simplicity, we assume that the set P is such that the partial satisfaction can be encoded with the $\vDash$ relation.

A *simulator* (or solver) $\mathcal{A}$ is an algorithm that computes the behavior trace of a dynamical model.

For a given model M and simulator $\mathcal{A}$, we denote the induced model by $[\![M]\!]_{\mathcal{A}}$. With this notation, the behavior trace computed is exact iff $[\![M]\!]_{\mathcal{A}} = M$, and approximate

otherwise.

In the case that $[\![M]\!]_{\mathcal{A}} \neq M$, we define the error of the simulator as the abstract difference between the original model and the induced model: $\|M - [\![M]\!]_{\mathcal{A}}\|$, for some given norm $\|\cdot\|$.

A simulator is accurate when $\|M - [\![M]\!]_{\mathcal{A}}\|$ is small enough to understand whether M satisfies the set of properties P.

Because we do not always have access to the behavior trace of the model, it is more realistic to redefine the notion of accuracy in terms of properties.
Definition 36. Given a set of properties P, a simulator $\mathcal{A}$ is accurate when it satisfies the same subset of properties as the model:

$$\forall p \in P, M \vDash p \Leftrightarrow [\![M]\!]_{\mathcal{A}} \vDash p \tag{8.1}$$

The measure of accuracy can then be the number of properties that satisfy Equation (8.1).
Remark 13. Definition 35 excludes performance related properties (e.g., execution time of the simulation algorithm). We argue that these are secondary properties whose satisfaction only makes sense when the primary properties P are satisfied (which means the simulation results can be trusted).

We use the term FMU to denote an executable artifact that produces a behavior trace, when inputs are provided. The FMU combines a simulation $\mathcal{A}$ with a model M, and produces the behavior trace of $[\![M]\!]_{\mathcal{A}}$.

A *simulation* is the behavior trace obtained with an FMU. The correctness of the simulation depends on the accuracy of the simulator (Definition 36) and the validity of the dynamical model (Definition 35).

A coupled model is a dynamical model that is comprised of sub-models. When the sub-models are represented by different FMUs, we need co-simulation to approximate the behavior trace of the coupled model.

A co-simulation is the behavior trace of a coupled model approximated by a master algorithm applied to a co-simulation scenario. A co-simulation scenario is a set of FMUs and their I/O mappings (e.g., see Figure 8.2). A master algorithm represents the approach to compute the co-simulation. It typically determines the communication rate, and which data is exchanged between FMUs. When an FMU represents a continuous sub-model, its inputs need to be approximated. As such, we consider the input approximation schemes as being part of the master algorithm.

In the following, we formalize the concepts of FMU, co-simulation scenario, and master algorithm, with the intent of exposing the nuances in configuring a co-simulation.

We adapt the notations introduced in [66]. To simplify and follow assumption 13, we leave out the notation for the initialization and feed-through. However, our implementation accounts for these omissions.
Definition 37. An FMU with identifier c is a structure $\langle S_c, U_c, Y_c, R_c, \texttt{set}_c, \texttt{get}_c, \texttt{doStep}_c \rangle$, where:

- S_c represents the state space;
- U_c and Y_c the set of input and output variables, respectively;
- $R_c : U_c \to \{true, false\}$ the reactivity of each input (see Definition 39);

- $\texttt{set}_c : S_c \times U_c \times \mathcal{V} \to S_c$ and $\texttt{get}_c : S_c \times Y_c \to \mathcal{V}$ are functions to set the inputs and get the outputs, respectively (we abstract the set of values that each input/output variable can take as $\mathcal{V}$); and
- $\texttt{doStep}_c : S_c \times \mathbb{R}_{\geq 0} \to S_c$ is a function that instructs the FMU to compute its state after a given time duration.

The following definition reflects the fact that the FMI Standard leaves implicit the current time of each FMU. However, this information is crucial to correctly configure the co-simulation.

Definition 38 (State timestamp)**.** Given a communication step size $H \in \mathbb{R}_{\geq 0}$ and $H > 0$, we say that the state $s_c \in S_c$ of an FMU c has timestamp t, denoted as $\varphi(s_c) = t$ when $\texttt{doStep}_c$ has been called $\frac{t}{H}$ times with H as parameter.

According to Definition 38, if an FMU is in state s_c at time t, $\texttt{doStep}_c(s_c, H)$ approximates the state of the corresponding model at time $t + H$. If this model is a continuous one, the FMU will approximate the evolution of the state in the interval $[t, t + H]$, using an approximation function to estimate the values of the inputs in that interval. In our notation, we choose to leave this function implicit in the $\texttt{doStep}_c$, as reflected in the current version of the FMI Standard. However, we make explicit the requirements of each kind of input approximation in the form of the reactivity R_c.

Intuitively, an FMU with a reactive input must wait until the FMU that feeds that input executes a step before getting that input value. The reactivity therefore imposes an order in the execution of the FMUs. This concept was first introduced in [158].

Definition 39 (Reactivity)**.** For a given FMU c with input $u \in U_c$, $R_c(u) = \mathit{true}$ if the function $\texttt{doStep}_c$ makes use of an interpolation of input u. Formally, let t be the timestamp of the state s_c prior to a call to $\texttt{doStep}_c(s_c, H)$, and let d denote the FMU whose output $y \in Y_d$ is connected to u. Then, $R_c(u) = \mathit{true}$ means that s_c must have been produced from a call to $\texttt{set}_c(\ldots, u, \texttt{get}_d(s_d, y))$ where the state s_d of FMU d satisfies $\varphi(s_d) = t + H$. Conversely, $R_c(u) = \mathit{false}$ means that s_c must have been produced from a call to $\texttt{set}_c(\ldots, u, \texttt{get}_d(s_d, y))$ where $\varphi(s_d) = t$.

Since knowing the reactivity of each FMU is related to having access to the input approximation implementation, and since the FMI Standard version 2.0 does not include information about reactivity, we make the following assumption.

Assumption 15. *If an FMU c does not disclose its input approximation scheme for an input u, then we assume that u is approximated with a constant extrapolation. Therefore, $R_c(u) = \mathit{false}$.*

Fortunately, the input approximation scheme of an FMU input, and therefore its reactivity, can be controlled by semantic adaptation.

Definition 40. *Semantic adaptation* is a technique that allows a new FMU c to be constructed from an old set of FMUs, using a custom implementation of the $\texttt{set}_c$,$\texttt{get}_c$, and $\texttt{get}_c$ functions (recall Chapter 7).

Definition 41. A co-simulation scenario is a structure $\langle C, L \rangle$ where each FMU identifier $c \in C$ is associated with an FMU, as defined in Definition 37, and $L(u) = y$ means that the output y is connected to input u. Let $U = \bigcup_{c \in C} U_c$ and $Y = \bigcup_{c \in C} Y_c$, then $L : U \to Y$.

A master algorithm is considered here as everything that influences the co-simulation result. The following concepts are a way to isolate and formalize these different components.

Definition 42 (Co-simulation Step)**.** Given a co-simulation scenario $\langle C, L\rangle$, a co-simulation step is an ordered sequence of FMU function calls $(f)_{i\in\mathbb{N}}$ with

$$f \in F = \bigcup_{c\in C} \{\texttt{set}_c, \texttt{get}_c, \texttt{doStep}_c\},$$

and i denoting the order of the function call. A function call f_i comes before a function call f_j, written as $f_i \twoheadrightarrow f_j$, if $i < j$, and comes immediately before, written as $f_i \rightarrow f_j$, if $i = j - 1$.

It is important that the co-simulation step respects the reactivity of each FMU (recall Definition 39), and the couplings of the FMUs.

Definition 43 (Valid Co-simulation Step)**.** Given a co-simulation step size $H > 0$, a co-simulation is valid with respect to reactivity and couplings if it satisfies the following conditions:

1. Each function call uses the most recent FMU State as parameter. For example, if $f_j = \texttt{get}_c(s_c, y)$ then s_c must be the result of the most recent call to $\texttt{set}_c$ or $\texttt{doStep}_c$, that is, the maximal i such that $i < j$, and $f_i = \texttt{set}_c(\ldots)$ or $f_i = \texttt{doStep}_c(\ldots)$.
2. For every $c \in C$, there exists one, and only one, call to $\texttt{doStep}_c$, and it is done with argument H.
3. Each call to $\texttt{doStep}_c$ for $c \in C$ must come after every call to $\texttt{set}_c$ on the input variables of c.
4. Each call to `get` is immediately followed by a sequence of calls to `set` to set the affected input variables.
5. For each $c \in C$ and $u \in U_c$ satisfying $R_c(u) = \mathit{true}$, $\texttt{doStep}_d \twoheadrightarrow \texttt{get}_d(L(u), \ldots)$, where $L(u) \in Y_d$ and $d \in C$.
6. For each $c \in C$ and $u \in U_c$ satisfying $R_c(u) = \mathit{false}$, the call to $\texttt{set}_c(\ldots, u)$ $\texttt{set}_c(\ldots, u) \twoheadrightarrow \texttt{doStep}_d$, where $L(u) \in Y_d$ and $d \in C$.

Remark 14. Regarding Definition 43:

- The most common master algorithms will satisfy conditions 1–3;
- Condition 4 is not mandatory but it facilitates the description of Conditions 5 and 6. Furthermore, it makes the implementation simpler.
- Conditions 5 and 6 ensure that the reactivity of each input is respected, according to Definition 39.
- This definition is consistent with assumption 13. Relaxing this assumption requires modifications that are outside the scope of this work.

In addition to Definition 43, we make the following assumption, which is not strictly required to ensure a valid co-simulation step, but makes the description of the techniques employed in later sections simpler.

Assumption 16. *In a valid co-simulation step there is only one call to* $\texttt{get}_d(y, \ldots)$

Assumption 16 restricts the reactivity of every two input variables u, v that are fed by the same output variable, that is $P(u) = P(v)$, to be the same. We do not lose generality by making this assumption since one can perform multiple calls to $\texttt{get}_d(y, \ldots)$, before and after $\texttt{doStep}_d$ to get the right values.

Executing a co-simulation step in a co-simulation scenario $\langle C, L\rangle$ where all FMUs $c \in C$ have a state s_c satisfying $\varphi(s_c) = t$, will update each FMU state s_c to satisfy $\varphi(s_c) = t+H$, where H is the argument of every call to `doStep`.

Definitions 42 and 43 and assumption 16 purposefully exclude the case where a group of FMUs needs to communicate more frequently per co-simulation step, as in experiment 2. This is because this group can be transformed to a single FMU using semantic adaptation (Definition 40). Therefore, we do not lose generality. We revisit this in Section 8.5.4.

Definition 44. Given a co-simulation scenario $\langle C, L \rangle$, a co-simulation step size H, and a co-simulation step $(f)_{i \in \mathbb{N}}$, a master algorithm is a structure defined as $\mathcal{A} = \left\langle C, L, H, (f)_{i \in \mathbb{N}} \right\rangle$.

With this formalization, we can summarize the configuration parameters of a master algorithm:

- co-simulation step size H;
- FMUs and their semantic adaptations C, L; and
- co-simulation step $(f)_{i \in \mathbb{N}}$.

Each different parameter induces a model that is likely to be different than the original coupled model.

Definition 45. The induced coupled model, denoted by $[\![M]\!]_{\mathcal{A}}$, is the model whose behavior trace is computed by a given master algorithm $\mathcal{A}$, with the intent of approximating the behavior trace of a coupled model M.

8.3.2 Research Problem

With these assumptions and definitions, we can formalize our main goal.

For a given coupled model M, find a master algorithm $\mathcal{A}$ that minimizes $\|M - [\![M]\!]_{\mathcal{A}}\|$.

Since in realistic conditions we do not have access to the behavior trace of M, we relax the above problem to the following.

Problem 3. *For a given set of properties P, a coupled model M, find a master algorithm*

$$\mathcal{A} = \left\langle C, L, H, (f)_{i \in \mathbb{N}} \right\rangle,$$

that maximizes the size of the set

$$\{p : p \in P, M \vDash p \implies [\![M]\!]_{\mathcal{A}} \vDash p\},$$

such that $(f)_{i \in \mathbb{N}}$ satisfies Definition 43 and assumption 16.

Intuitively, the solution $\mathcal{A}$ to Problem 3 can be seen as having maximal relaxed accuracy (recall Definition 36).

Multiple solutions to Problem 3 are possible. In this work, we provide a way to generate multiple potential solutions for the user to evaluate.

Note that if the coupled model M is invalid (recall Definition 35), the optimal solution $\mathcal{A}$ would have to not satisfy the same properties that the model does not satisfy. Therefore, we make the following assumption:

Assumption 17. *When solving Problem 3, we assume that M is valid, according to Definition 35.*

As well, in practice the set of properties P is not completely specified. This motivates our proposal of using *hints* as an approximation of P, to be derived from requirements, or declared by engineers. Assumption 18 reflects that we must rely on these hints to obtain information about M.

```
Hint ExecRate{
    description "Controller FMU is software."
    statements {
        Property ExecRate :=
            FMUProperty FMU1.exec_rate == val 1.0e+6 hz
    }
    scope Globally
    pattern Universality:always-the-case-that ExecRate holds
}
Hint PowerBond{
    description "Plant/Load FMUs share a power connection."
    statements {
        Property PowerBond :=
            Plant.f == PowerBondSuggestions with Load.v
    }
    scope Globally
    pattern Universality:always-the-case-that PowerBond holds
}
```

Figure 8.6: The *ExecRate* and *PowerBond* hints.

Assumption 18. *We assume that M's behavior satisfies the hints provided.*

In the next section, we describe how these hints are represented such that the user does not need to understand the co-simulation domain. Then, in Section 8.5 we discuss our approach to solve Problem 3.

8.4 Hint Language

In this section, we describe how to represent the hints used to configure co-simulation as defined in Problem 3. This is done through the creation of a small *domain-specific language* (DSL). DSLs allow experts in the problem space (the system engineers) to describe hints, without having to become experts in the solution space (the co-simulation domain) [368].

As an example, Figure 8.6 show the hints described in assumption 12. Each hint has a number of fields. The *description* field is an unstructured text, as commonly seen in industrial requirements. Following this are *statements*, which can be *events* or *properties*. Finally, the *scope* and the *pattern* specify when the hint is valid.

Statements As seen in Figure 8.6, *Statements* define the *Events* and *Properties* which refer to FMUs and their signals in the system.

For brevity, we omit the description of several other operators that can be used as statements. For example, hints can be specified over the average or derivative of a signal. The language is defined to be easily extensible, and we are collaborating with our industrial partners to define further useful operators.

Scopes and Patterns The example hints in Figure 8.6 are applicable throughout the entire simulated time. This is denoted by the *Globally scope* of the hint, and the *Universality pattern*. These scopes and patterns are sourced from [29].

Scopes define when the hint is valid: *Globally*, *Before an event*, *After an event*, *Between two events*, and *After an event until another event*.

Patterns define the precise manner in which a statement holds. We refer the reader to [29] for a full description and syntax of the patterns. As an example, the *MinDuration* pattern means *once [an event] holds, it will hold for [an amount of time]*. This allows the engineer to precisely define how the statements defined in the hint should hold.

Implementation The hint language was created with the XText DSL framework[1], which produces a XML Metadata Interchange (XMI) file representing each hint. These files are then used for the generation of the master algorithms that are candidate solutions to Problem 3.

8.5 Master Generation

This section describes the search algorithm that enumerates potential solutions to Problem 3 using a given set of hints.

8.5.1 Search Space Representation

Given a co-simulation scenario, the search space is the set of all master algorithms that can be used to compute the co-simulation. According to Definition 44, we identify the following dimensions of the search space:

- the set of all communication step sizes;
- the set of all co-simulation steps;
- the set of all semantic adaptations applied to the FMUs.

The search space is therefore infinite, though as shown below we consider only a finite subset of this space.

The problem of representing a search space is not new, and there is a rich literature in design space exploration from where we draw inspiration. In particular, we borrow concepts from feature models [202] for the representation, and the use of domain-specific hints for optimization (e.g., [364, 370]).

Figure 8.7 shows an excerpt metamodel of the search space representation. The search space, represented by class *Candidates*, comprises multiple *RootCandidateScenarios*. The latter represent alternative master algorithms. These comprise the co-simulation *step size*, the co-simulation *unit instances* (FMUs), their *connections*, and *semantic adaptations*. The co-simulation step operations are left implicit, restricted by the semantic adaptations used (see Section 8.5.3).

Semantic adaptations can be applied to FMUs and input ports, and since FMUs can be hierarchical, this enables the representation of multi-rate master algorithms. Of the currently supported semantic adaptations, we highlight:

Extrapolation/Interpolation Adaptation Applies the approximation to the affected FMU input port.

MultiRateAdaptation Makes an FMU perform multiple steps per co-simulation step. Can be combined with *Extrapolation/Interpolation* adaptations, and can also be applied to hierarchical FMUs.

PowerBond Adaptation Whenever two FMUs share a power connection, the *PowerBond* adaptation changes one of the FMU's input ports to correct for the energy dissipated, using the technique introduced in [43].

XOR Adaptation Can be combined with other adaptations to represent alternatives.

[1] `https://www.eclipse.org/Xtext/`

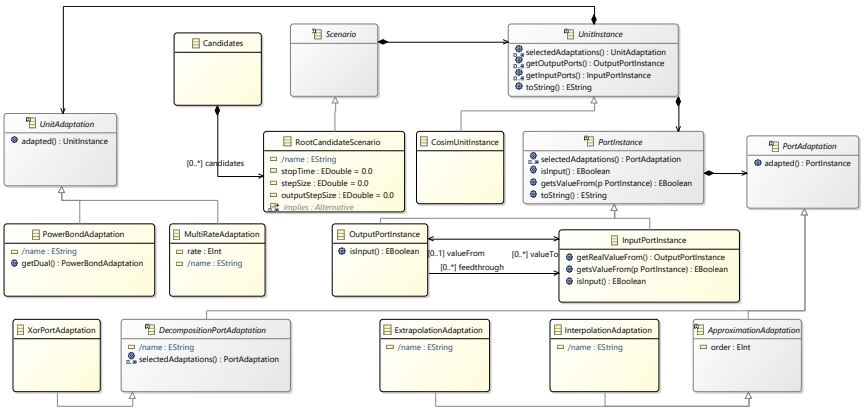


Figure 8.7: Excerpt of search space representation metamodel.

The variability of the co-simulation candidates is therefore encoded in alternative root candidate scenarios and *XorAdaptations*.

Example 26. Figure 8.8 shows an example search space. The `Load` and `Plant` FMUs have a *PowerBond* adaptation, and the `Environment` FMU has an *XorAdaptation* with two alternative multi-rate adaptations. This search space represents four alternative master algorithms, because of the two `Environment` FMU rates, represented as $R = \{100, 10\}$ in the figure, and two possible communication step sizes, represented as $H = \{1 \times 10^{-7}, 1 \times 10^{-6}\}$.

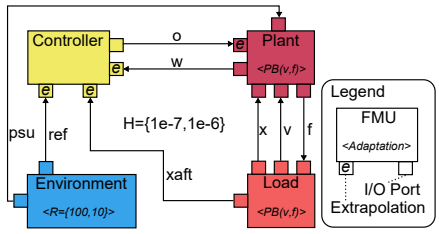


Figure 8.8: Example search space.

Given a set of hints, our tool generates a search space representation, according to the algorithm described in Section 8.5.5. In order to explain the algorithm, we first have to describe how different *master algorithm variants* are generated and executed.

8.5.2 Variant Generation

Definition 46. Given a search space representation, a *master algorithm variant* represents a set of decisions made about each variation point represented in the search space.

Definition 47. A *variant diagram* represents all possible variants in the search space. It is a rooted, connected, directed, and acyclic graph, where each node has the same children as every other sibling node. Each node is associated with a weight and represents a decision relevant to the master algorithm. The children of the same node represent a set of mutually exclusive choices, where the weight of each child represents the preference for each choice.

The higher the weight of a node, the closer to the root that node and its siblings will be.
Example 27. Figure 8.9 shows a variant diagram representing the variants encoded in Example 26. The co-simulation step nodes are just below the root because the co-simulation step 1×10^{-7} s has the maximal weight of the whole tree.

For now, we assume for brevity that these co-simulation scenarios do not have hierarchical FMUs. This assumption is relaxed in Section 8.5.4.

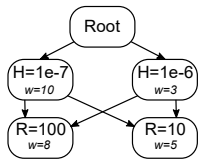


Figure 8.9: An example variant diagram.

Definition 48. Given a variant diagram, a master algorithm variant is defined as being a path from the root node to a leaf node.

The number of possible variants in a diagram is the number of different paths from the root to a leaf node. Since it is not feasible to try all variants for a large co-simulation scenario, we rank them.

The variants are ranked according to the weight on each node, using the search procedure defined below. These weights are set by the procedure that generates the search space from the hints (Section 8.5.5).

---

1 Starting at the root, each child node with the highest weight is visited first. A variant is created when the search reaches a leaf of the tree. The search then continues in the tree using backtracking, heading up to the closest sibling node with the highest weight (among siblings) not yet chosen.

Example 28. Line 14 generates the following variants, in order: $\langle H = 1 \times 10^{-7}, R = 100 \rangle$; $\langle H = 1 \times 10^{-7}, R = 10 \rangle$; $\langle H = 1 \times 10^{-6}, R = 100 \rangle$; and $\langle H = 1 \times 10^{-6}, R = 10 \rangle$.

Note that that the weight value is assigned by our system, but is only valid to select between child nodes. That is, weights cannot be compared between branches of the tree, and a path's weight cannot be compared to another path.

As well, currently the weights are statically assigned when the diagram is created. A future implementation may be to dynamically assign weights based on the decisions already taken.

8.5.3 Variant Execution

A variant encodes the co-simulation scenario, semantic adaptations to be applied, and parameter values. A master algorithm comprises the co-simulation step, as defined in Definitions 42 and 44, which needs to satisfy the conditions in Definition 43 and assumption 16, which in turn depend on the semantic adaptations selected for the variant. For example, an input that is bound to a interpolation adaptation must be reactive (recall Definition 39). In order to represent all the constraints in the execution order of operations, we introduce the following structure.

Definition 49. Given a variant, we define the corresponding operation schedule as a directed graph where each node represents an operation in F (Definition 42), and each edge between nodes i and j means that the operation represented by i must be executed before the operation represented by node j. The edges are created according to the semantic adaptations selected for each unit and port, as described in Definition 43.
Example 29. Figure 8.10 shows the operation schedule of all variants described in Example 27.

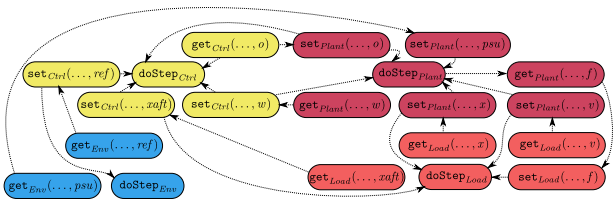


Figure 8.10: Example operation schedule for variants in Figure 8.9. The edges represent ordering constraints, as in Definition 49. A possible topological order is displayed on the left.

Definition 50. A variant is executable if the corresponding operation schedule has a topological sort.

Furthermore, according to the assumptions we have made, if there are multiple topological orderings, they are all behaviorally equivalent.

8.5.4 Hierarchical FMUs

To keep the explanation simple, we omitted how hierarchical FMUs are handled. Since a Hierarchical FMU represents essentially a co-simulation scenario, they are treated the same way when it comes to variant representation and generation. The main difference lies in the creation of the operation schedule.

---

1 Given a variant that contains hierarchical FMUs, we build the operation schedule according to Definition 49, but excluding the operations that correspond to child FMUs of the hierarchical FMUs. Then, we recursively create the operation schedule of the FMUs inside each hierarchical FMU. The operation schedule of each Hierarchical FMU c is run whenever the $\texttt{doStep}_c$ operation is invoked.

8.5.5 Search Space Generation

The previous sections cover the generation and execution of variants once the search space is defined. In this section, we focus on how the search space is created. To be succinct, we will only focus on the hints that were applied to the case study.

8.5.6 Results

We applied the above procedure to the co-simulation scenario introduced in our motivating example in Section 8.2, with the *ExecRate* and *PowerBond* hints described in Section 8.4.

1 Given a set of hints and a co-simulation scenario, the search space is created as follows:
1. For each FMU with a software controller hint, add an extrapolation adaptation to each of its input ports and to each of the input ports that are connected to its outputs.
2. If there are multiple software hints with different configured frequency rates, define the scenario step size to be the inverse of the minimum of the frequency rates, and define the appropriate multi-rate adaptations on the software FMUs.
3. For each PowerBond hint, add a power bond adaptation to each of the FMUs sharing the bond.
4. Select the FMUs that are not affected by any hint, and add a multi-rate adaptation (if not already defined) with alternative step sizes at different orders of magnitude, and two alternative first order input approximations (interpolation and extrapolation). Higher weights are given to smaller step sizes and interpolations.
5. If the co-simulation step has not been defined yet, define multiple alternative co-simulation steps with different orders of magnitude. Higher weights are given to smaller step sizes.

This resulted in the search space in Figure 8.8, and the four possible master algorithms, described in Example 28.

All variants produced a smooth signal, shown in Figure 8.11. The fastest variant took 6 minutes, on average, to produce the result. The slowest variant took 38 minutes.

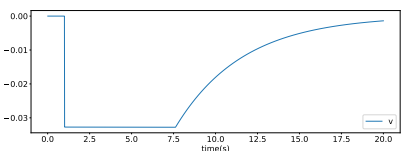


Figure 8.11: Co-simulation computed from the hints provided in Figure 8.6, with the variant $\langle H = 1 \times 10^{-6}\,\mathrm{s}, R = 10 \rangle$.

8.6 Related Work

The problem of adequately configuring a co-simulation is not new. We can classify the approaches in two categories: adaptive and static configuration.

In the adaptive category, we highlight the adaptive co-simulation schemes:

Input Approximation/Correction [39, 40, 43, 71, 116, 161]; and
Step-size Correction [24, 26, 48, 66, 73, 75, 136, 184, 317]

Our approach complements the state of the art in providing a way to select which methods are best applicable. More importantly, our approach acknowledges that the same technique cannot be applied to every FMU in the co-simulation scenario. For example, the energy correction scheme in [161] works very well, but requires each FMU to provide an energy function. Our work can be used to describe which signals represent the energy function.

In the static configuration category, the following works have the same goal as this paper.

The authors in [214] propose to use system models to configure the co-simulation. However, their approach to configure the co-simulation differs from ours by not attempting to generate multiple candidate master algorithms. They also focus only on ensuring the configuration is syntactically correct (e.g., the FMU connections are not consistent with the model). In principle, it is possible to adapt their proposed language to include hints and explore multiple good master algorithms.

The work in [41] recognizes the need to take into account the input/output feed-through, and the kind of model underlying the FMU, in order to configure the co-simulation. It shows that the input approximation techniques and the step size adaptation can interfere with each other. We complement this work by showing that there is more information that can be used to configure the co-simulation. Additionally, we acknowledge that multiple master algorithms can perform well, instead of focusing on generating just one.

The approach in [185] is similar to our own, as it formulates an optimization problem to find the best co-simulation step (as defined in Definition 42). However, in contrast with our work, they make the assumption that FMUs can perform interpolation or extrapolations equally well. We make the distinction between reactive and delayed inputs. As well, we formulate an optimization problem whose solution attempts to satisfy the hints provided by the user. In contrast, they assume that it is always better to have interpolations over extrapolations. For example, as we discussed in Section 8.2, a software controller FMU should not be forced to make interpolations, as that violates causality. In the future, it would be interesting to incorporate the contribution in [185] to solve the sequence ordering sub-problem, within the larger optimization loop of finding a master algorithm that satisfies the given hints.

8.7 Concluding Remarks

Due to the circumstances in which co-simulation is applied, there are often no analytical results to guide the search for a correct configuration of the co-simulation.

This chapter represents a first step to this goal, by presenting a framework that allows users to explore various options for configuring their co-simulation scenarios, using hints. These hints are used to produce a search plan for the different co-simulation variants, taking into account the best practices in the state of the art. Each variant is then evaluated and the results are presented to the user.

Limitations and Assumptions. A major assumption that we have made is that the coupled model being co-simulated satisfies the hints described by the engineers (assumptions 17 and 18). However, it is often the case that the users of co-simulation are trying to understand the model and therefore may not describe correct hints. Our contribution can help by showing whether there is agreement between master algorithms on the satisfiability of a particular set of hints. A hint that is not satisfied by any of the co-simulation algorithms generated suggests that perhaps the problem is in the model and not in the co-simulation.

Ongoing work is focused on supporting more hints, including proprietary hints. In the future, we intent to formalize some of the hints in Signal Temporal Logic, and compute a degree of satisfaction per hint (Remark 12). This will allow us to use global optimization techniques to improve the recommended master algorithms. Another direction is to derive

the hints from prior experience with the components. For example, descriptions of previous physical experiments, in the form of experimental frames [109], could be used.

For Boeing, the candidate master algorithms suggested by the HintCO tool represent a first step towards enabling seamless integrated large scale simulation of heterogeneous models. However, there are still challenges that need to be overcome. For example, tool vendors need to improve the maturity level of FMU support, to focus on enhanced cross compatibility and integration checks (assumption 13). Furthermore, there is a need to develop a common methodology and modeling guidelines for industrial applications.

Chapter 9

Conclusion

Co-simulation tackles the challenges originating from concurrency and specialization in the development of complex systems. However, the characteristics that allow the technique this privilege, are also the ones that aggravate the fundamental challenge in simulation based analyses: deciding whether the results can be trusted.

This work represents a first step towards allowing co-simulation to seamlessly become integral part of existing development processes. We have studied the state of the art, and queried practitioners for their views. Moreover, we have advanced our knowledge by tackling some of the challenges identified.

In order to understand the state of the art, we performed a systematic literature survey, which was then used to formulate the questions for an empirical survey. The conclusions draw from these studies guided our subsequent contributions in the stability analysis, and configuration, of co-simulations.

The main limitation of our contributions in stability of co-simulations (Chapters 5 and 6), is that they are not applicable to more complex models. As a first step, simple models, for which we have the analytical simulation. These models allow one to understand the trade-offs involved, the limitations of each contribution, and provide good benchmark models for future research work. More fundamental research is needed to make these techniques applicable to more complex models. For now, one can use abstract simplified models to apply the analyses, and draw hypotheses about the co-simulation of more complex models. These can then be corroborated with experiments.

In practice, as we show later in Chapters 7 and 8, practitioners seldom have access to the analytical behavior of the system. We worked around this problem by showing which information can be used to improve the chances of producing correct results, without sacrificing the Intellectual Property in the models. This information concerns implementation details of each simulator, and domain knowledge regarding the expected behavior of the system.

Recognizing that the challenges we tackled are far from trivial, each technical contribution attempts to rigorously formulate the problem we proposed to solve, discusses the limitations of our solution, and highlights potential research directions. As an example, both

stability analyses techniques that we develop highlight challenges that are not specific to co-simulation: the stability analysis of adaptive co-simulations can also be used to study the stability of consensus algorithms; and the stability analysis of hybrid co-simulations can be used to study the stability of network controllers.

It is our hope that this work will help practitioners using co-simulation, and entice researchers of every background to work in this exciting field.

Bibliography

[1] *ACOSAR: Research Project*. Cited on page 28.

[2] *COSIBAS: Research project*. Cited on page 28.

[3] *CyDER: Research Project*. Cited on page 28.

[4] *DESTECS: Research Project*. Cited on page 28.

[5] *EMPHYSIS: Research Project*. Cited on page 28.

[6] *ERIGrid: Research Project*. Cited on page 28.

[7] *INTO-CPS: Reseach Project*. Cited on page 28.

[8] *MODELISAR: Research Project*. Cited on page 28.

[9] *ODETTE: Research Project.* Cited on page 28.

[10] *OpenCPS: Research Project*. Cited on page 28.

[11] *PEGASUS: Research Project*. Cited on page 28.

[12] *Xtext - Language Engineering for Everyone*. Cited on page 167.

[13] *Modelica - A Unified Object-Oriented Language for Physical Systems Modeling*, 2007. Pages: Version 3.0. Cited on page 49.

[14] *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification*, 2010. Cited on pages 18, 38, and 64.

[15] M. ADLER AND E. ZIGLIO, *Gazing Into the Oracle: The Delphi Method and Its Application to Social Policy and Public Health*, Jessica Kingsley Publishers, London and Philadelphia, 1996. Cited on pages 74 and 75.

[16] A. A. AHMADI, R. JUNGERS, P. A. PARRILO, AND M. ROOZBEHANI, *Analysis of the joint spectral radius via lyapunov functions on path-complete graphs*, in Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control - HSCC '11, Chicago, IL, USA, 2011, ACM Press, p. 13. Cited on page 113.

[17] R. ALUR, C. COURCOUBETIS, N. HALBWACHS, T. A. HENZINGER, P. H. HO, X. NICOLLIN, A. OLIVERO, J. SIFAKIS, AND S. YOVINE, *The algorithmic analysis of hybrid systems*, Theoretical Computer Science, 138 (1995), pp. 3–34. Cited on page 60.

[18] A. A. ALVAREZ CABRERA, K. WOESTENENK, AND T. TOMIYAMA, *An architecture model to support cooperative design for mechatronic products: A control design case*, Mechatronics, 21 (2011), pp. 534–547. Cited on pages 2 and 19.

[19] C. ANDERSSON, *Methods and Tools for Co-Simulation of Dynamic Systems with the Functional Mock-up Interface*, PhD thesis, Lund University, 2016. Series Title: Doctoral Theses in Mathematical Sciences. Cited on pages 43, 58, 63, and 144.

[20] C. ANDERSSON, C. FÜHRER, AND J. ÅKESSON, *Efficient Predictor for Co-Simulation with Multistep Sub-System Solvers*, tech. rep., Centre for Mathematical Sciences, Lund University, 2016. Publication Title: Technical Report in Mathematical Sciences. Cited on pages 58, 63, and 146.

[21] E. P. ANDERT JR. AND D. MORGAN, *Collaborative Virtual Prototyping and Test*, Naval Engineers Journal, 110 (1998), pp. 17–23. Cited on page 17.

[22] F. ARBAB, I. HERMAN, AND P. SPILLING, *An Overview of Manifold and Its Implementation*, Concurrency: Pract. Exper., 5 (1993), pp. 23–70. Cited on page 17.

[23] M. ARNOLD, *Stability of Sequential Modular Time Integration Methods for Coupled Multibody System Models*, Journal of Computational and Nonlinear Dynamics, 5 (2010), p. 9. Cited on pages 43, 45, 47, 51, 53, 54, 57, 83, 91, 144, 174, and 179.

[24] M. ARNOLD, C. CLAUSS, AND T. SCHIERZ, *Error Analysis and Error Estimates for Co-simulation in FMI for Model Exchange and Co-Simulation v2.0*, in Progress in Differential-Algebraic Equations, Berlin, Heidelberg, 2014, Springer Berlin Heidelberg, pp. 107–125. Cited on pages 51, 53, 83, 116, 138, 140, 174, 179, 183, and 194.

[25] M. ARNOLD AND M. GÜNTHER, *Preconditioned Dynamic Iteration for Coupled Differential-Algebraic Systems*, BIT Numerical Mathematics, 41 (2001), pp. 1–25. Cited on pages 18, 19, 45, 47, 51, 53, and 54.

[26] M. ARNOLD, S. HANTE, AND M. A. KÖBIS, *Error analysis for co-simulation with force-displacement coupling*, PAMM, 14 (2014), pp. 43–44. Cited on pages 53, 174, and 194.

[27] M. ASLAN, U. DURAK, AND K. TAYLAN, *MOKA: An Object-Oriented Framework for FMI Co-Simulation*, in Conference on Summer Computer Simulation, Chicago, Illinois, July 2015, Society for Computer Simulation International San Diego, CA, USA, pp. 1–8. Cited on page 177.

[28] K. J. ASTRÖM AND R. M. MURRAY, *Feedback Systems: An Introduction for Scientists and Engineers*, Princeton university press, 2010. Cited on page 39.

[29] M. AUTILI, L. GRUNSKE, M. LUMPE, P. PELLICCIONE, AND A. TANG, *Aligning Qualitative, Real-Time, and Probabilistic Property Specification Patterns Using a Structured English Grammar*, IEEE Transactions on Software Engineering, 41 (2015), pp. 620–638. Cited on page 189.

[30] M. U. AWAIS, W. MUELLER, A. ELSHEIKH, P. PALENSKY, AND E. WIDL, *Using the HLA for Distributed Continuous Simulations*, in 8th EUROSIM Congress on Modelling and Simulation, Cardiff, UK, Sept. 2013, IEEE, pp. 544–549. Cited on pages 19 and 64.

[31] M. U. AWAIS, P. PALENSKY, A. ELSHEIKH, E. WIDL, AND S. MATTHIAS, *The high level architecture RTI as a master to the functional mock-up interface components*, in International Conference on Computing, Networking and Communications, San Diego, USA, Jan. 2013, IEEE, pp. 315–320. Cited on pages 19, 61, and 146.

[32] M. U. AWAIS, P. PALENSKY, W. MUELLER, E. WIDL, AND A. ELSHEIKH, *Distributed hybrid simulation using the HLA and the Functional Mock-up Interface*, in IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society, Vienna, Austria, Nov. 2013, IEEE, pp. 7564–7569. Cited on page 19.

[33] R. BALASUBRAMANIAN AND D. AGARWAL, *Delphi Technique- A Review*, International Journal of Public Health Dentistry, 3 (2012), pp. 16–25. Cited on page 74.

[34] F. J. BARROS, *Modeling formalisms for dynamic structure systems*, ACM Transactions on Modeling and Computer Simulation, 7 (1997), pp. 501–515. Place: New York, NY, USA. Cited on pages 37 and 64.

[35] J. BASTIAN, C. CLAUSS, S. WOLF, AND P. SCHNEIDER, *Master for Co-Simulation Using FMI*, in 8th International Modelica Conference, Dresden, Germany, June 2011, Linköping University Electronic Press, Linköpings universitet, pp. 115–120. Cited on pages 45, 51, 138, 174, 182, and 183.

[36] N. W. BAUER, P. J. H. MAAS, AND W. P. M. H. HEEMELS, *Stability analysis of networked control systems: A sum of squares approach*, Automatica, 48 (2012), pp. 1514–1524. Cited on page 135.

[37] G. BELTRAME, D. SCIUTO, AND C. SILVANO, *Multi-Accuracy Power and Performance Transaction-Level Modeling*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 26 (2007), pp. 1830–1842. Cited on pages 19 and 88.

[38] T. BELYTSCHKO, H.-J. YEN, AND R. MULLEN, *Mixed methods for time integration*, Computer Methods in Applied Mechanics and Engineering, 17-18 (1979), pp. 259–275. Cited on page 16.

[39] A. BEN KHALED, L. DUVAL, M. E. M. B. GAÏD, AND D. SIMON, *Context-based polynomial extrapolation and slackened synchronization for fast multi-core simulation using FMI*, in 10th International Modelica Conference, Lund, Sweden, 2014, Linköping University Electronic Press, pp. 225–234. Cited on pages 43, 174, and 194.

[40] A. BEN KHALED-EL FEKI, L. DUVAL, C. FAURE, D. SIMON, AND M. BEN GAID, *CHOPtrey: Contextual online polynomial extrapolation for enhanced multi-core co-simulation of complex systems*, SIMULATION, 93 (2017). Cited on pages 43, 184, and 194.

[41] M. BENEDIKT AND F. R. HOLZINGER, *Automated configuration for non-iterative co-simulation*, in 17th International Conference on Thermal, Mechanical and Multi-Physics Simulation and Experiments in Microelectronics and Microsystems (EuroSimE), Montpellier, Apr. 2016, IEEE, pp. 1–7. Cited on page 195.

[42] M. BENEDIKT, D. WATZENIG, J. ZEHETNER, AND A. HOFER, *Macro-step-size selection and monitoring of the coupoling error for weak coupled subsystems in*

the frequency-domain, V International Conference on Computational Methods for Coupled Problems in Science and Engineering, (2013), pp. 1–12. Cited on page 83.

[43] M. BENEDIKT, D. WATZENIG, J. ZEHETNER, AND A. HOFER, *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, Ibiza, Spain, June 2013, pp. 1–12. Cited on pages 184, 190, and 194.

[44] A. BENVENISTE, B. CAILLAUD, AND P. LE GUERNIC, *Compositionality in Dataflow Synchronous Languages: Specification and Distributed Code Generation*, Information and Computation, 163 (2000), pp. 125–171. Cited on page 51.

[45] C. BERTSCH, E. AHLE, AND U. SCHULMEISTER, *The Functional Mockup Interface-seen from an industrial perspective*, in 10th International Modelica Conference, 2014. Cited on pages 1 and 19.

[46] R. BJORNSON, N. CARRIERO, D. GELERNTER, T. MATTSON, D. KAMINSKY, AND A. SHERMAN, *Experience with linda*, Yale University Computer Science Department, Technical Report RR-866, (1991). Cited on page 17.

[47] T. BLOCHWITZ, M. OTTER, M. ARNOLD, C. BAUSCH, C. CLAUSS, H. ELMQVIST, A. JUNGHANNS, J. MAUSS, M. MONTEIRO, T. NEIDHOLD, D. NEUMERKEL, H. OLSSON, J.-V. PEETZ, AND S. WOLF, *The Functional Mockup Interface for Tool independent Exchange of Simulation Models*, in 8th International Modelica Conference, Dresden, Germany, June 2011, Linköping University Electronic Press; Linköpings universitet, pp. 105–114. Cited on pages 1, 19, 59, and 83.

[48] T. BLOCKWITZ, M. OTTER, J. AKESSON, M. ARNOLD, C. CLAUSS, H. ELMQVIST, M. FRIEDRICH, A. JUNGHANNS, J. MAUSS, D. NEUMERKEL, H. OLSSON, AND A. VIEL, *Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models*, in 9th International Modelica Conference, Munich, Germany, Nov. 2012, Linköping University Electronic Press, pp. 173–184. Cited on pages 49, 52, 54, 62, 64, 66, 141, 183, and 194.

[49] V. D. BLONDEL AND J. N. TSITSIKLIS, *The boundedness of all products of a pair of matrices is undecidable*, Systems & Control Letters, 41 (2000), pp. 135–140. Cited on pages 97, 102, and 121.

[50] B. W. BOEHM, C. ABTS, A. W. BROWN, S. CHULANI, B. K. CLARK, E. HOROWITZ, R. MADACHI, D. J. REIFER, AND B. STEECE, *Software Cost Estimation with Cocomo II*, Prentice Hall, 2000. Cited on page 173.

[51] BOEING, *Developing Airplane Systems Faster and with Higher Quality through Model-Based Engineering*, May 2017. Cited on page 180.

[52] S. BOGOMOLOV, M. GREITSCHUS, P. G. JENSEN, K. G. LARSEN, M. MIKUČIONIS, T. STRUMP, AND S. TRIPAKIS, *Co-Simulation of Hybrid Systems with SpaceEx and Uppaal*, in 11th International Modelica Conference, Paris, France, Sept. 2015, Linköping University Electronic Press, pp. 159–169. Cited on page 64.

[53] J.-S. BOLDUC AND H. VANGHELUWE, *Expressing ODE models as DEVS: Quantization approaches.*, in AI, Simulation and Planning in High Autonomy Systems, Lisbon, Portugal, 2002, IEEE, pp. 163–169. Cited on pages 61 and 146.

[54] ——, *Mapping ODES to DEVS: Adaptive quantization*, in Summer Computer Simulation Conference, Montreal, Quebec, Canada, July 2003, Society for Computer Simulation International, pp. 401–407. Cited on pages 61, 62, and 146.

[55] M. BOMBINO AND P. SCANDURRA, *A model-driven co-simulation environment for heterogeneous systems*, International Journal on Software Tools for Technology Transfer, 15 (2013), pp. 363–374. Cited on page 62.

[56] S. BOSCHERT AND R. ROSEN, *Digital Twin—The Simulation Aspect*, in Mechatronic Futures, Springer International Publishing, Cham, 2016, pp. 59–74. Cited on page 20.

[57] F. BOUCHHIMA, M. BRIÈRE, G. NICOLESCU, M. ABID, AND E. ABOULHAMID, *A SystemC/Simulink Co-Simulation Framework for Continuous/Discrete-Events Simulation*, in IEEE International Behavioral Modeling and Simulation Workshop, IEEE, Sept. 2006, pp. 1–6. Cited on page 62.

[58] O. BOUISSOU, A. CHAPOUTOT, AND A. DJOUDI, *Enclosing Temporal Evolution of Dynamical Systems Using Numerical Methods*, in NASA Formal Methods, Moffett Field, CA, USA, 2013, Springer Berlin Heidelberg, pp. 108–123. Cited on pages 22 and 83.

[59] F. BOULANGER, A. DOGUI, C. HARDEBOLLE, C. JACQUET, D. MARCADET, AND I. PRODAN, *Semantic Adaptation Using CCSL Clock Constraints*, in Models in Software Engineering SE - 12, J. Kienzle, ed., vol. 7167, Springer Berlin Heidelberg, 2012, pp. 104–118. Series Title: Lecture Notes in Computer Science. Cited on page 137.

[60] F. BOULANGER AND C. HARDEBOLLE, *Simulation of Multi-Formalism Models with ModHel'X*, in 1st International Conference on Software Testing, Verification, and Validation, Lillehammer, Norway, 2008, IEEE Computer Society, pp. 318–327. Cited on pages 19 and 176.

[61] F. BOULANGER, C. HARDEBOLLE, C. JACQUET, AND D. MARCADET, *Semantic Adaptation for Models of Computation*, in 11th International Conference on Application of Concurrency to System Design (ACSD), Newcastle Upon Tyne, UK, 2011, IEEE, pp. 153–162. Cited on pages 61, 64, 137, and 176.

[62] M. BRANICKY, *Multiple Lyapunov functions and other analysis tools for switched and hybrid systems*, IEEE Transactions on Automatic Control, 43 (1998), pp. 475–482. Cited on page 116.

[63] M. S. BRANICKY, V. S. BORKAR, AND S. K. MITTER, *A unified framework for hybrid control: Model and optimal control theory*, IEEE Transactions on Automatic Control, 43 (1998), pp. 31–45. Cited on page 63.

[64] J. BREMBECK, A. PFEIFFER, M. FLEPS-DEZASSE, M. OTTER, K. WERNERSSON, AND H. ELMQVIST, *Nonlinear State Estimation with an Extended FMI 2.0 Co-Simulation Interface*, in 10th International Modelica Conference, Lund, Sweden,

Mar. 2014, Linköping University Electronic Press; Linköpings universitet, pp. 53–62. Cited on pages 43 and 174.

[65] D. BROMAN, *Hybrid Simulation Safety: Limbos and Zero Crossings*, in Principles of Modeling, vol. 10760, Springer International Publishing, Cham, 2018, pp. 106–121. Cited on pages 63 and 83.

[66] D. BROMAN, C. BROOKS, L. GREENBERG, E. A. LEE, M. MASIN, S. TRIPAKIS, AND M. WETTER, *Determinate composition of FMUs for co-simulation*, in Eleventh ACM International Conference on Embedded Software, Montreal, Quebec, Canada, 2013, IEEE Press Piscataway, NJ, USA, p. Article No. 2. Cited on pages 44, 51, 54, 62, 176, 185, and 194.

[67] D. BROMAN, L. GREENBERG, E. A. LEE, M. MASIN, S. TRIPAKIS, AND M. WETTER, *Requirements for Hybrid Cosimulation Standards*, in 18th International Conference on Hybrid Systems: Computation and Control, Seattle, Washington, 2015, ACM New York, NY, USA, pp. 179–188. Series Title: HSCC '15. Cited on pages 30, 62, 64, and 176.

[68] J. T. BUCK, S. HA, E. A. LEE, AND D. G. MESSERSCHMITT, *Ptolemy: A framework for simulating and prototyping heterogeneous systems*, International Journal of Computer Simulation, 4 (1994), pp. 155–182. Cited on page 64.

[69] R. L. BURDEN AND J. D. FAIRES, *Numerical Analysis*, Cengage Learning, 9 ed., 2010. Cited on pages 145, 146, and 174.

[70] M. BUSCH, *Zur Effizienten Kopplung von Simulationsprogrammen*, PhD thesis, Kassel university, Germany, 2012. Cited on pages 146 and 174.

[71] ——, *Continuous approximation techniques for co-simulation methods: Analysis of numerical stability and local error*, Journal of Applied Mathematics and Mechanics, 96 (2016), pp. 1061–1081. Cited on pages 43, 51, 57, 58, 83, 91, 95, 97, 110, 113, 138, 144, 146, 174, and 194.

[72] M. BUSCH AND B. SCHWEIZER, *Numerical stability and accuracy of different co-simulation techniques: Analytical investigations based on a 2-DOF test model*, in 1st Joint International Conference on Multibody System Dynamics, 2010, pp. 25–27. Cited on pages 55, 57, 83, and 91.

[73] ——, *An explicit approach for controlling the macro-step size of co-simulation methods*, in 7th European Nonlinear Dynamics, Rome, Italy, 2011, European Mechanics Society, pp. 24–29. Cited on pages 45, 53, 54, 83, 174, and 194.

[74] ——, *Stability of Co-Simulation Methods Using Hermite and Lagrange Approximation Techniques*, in ECCOMAS Thematic Conference on Multibody Dynamics, Brussels, Belgium, July 2011, pp. 1–10. Cited on pages 57 and 91.

[75] ——, *Coupled simulation of multibody and finite element systems: An efficient and robust semi-implicit coupling approach*, Archive of Applied Mechanics, 82 (2012), pp. 723–741. Cited on pages 54, 174, 183, and 194.

[76] J. BUUR ET AL., *Mechatronics Design in Japan*, PhD thesis, Institute for Engineering Design, Technical University of Denmark (DTH), 1989. Cited on pages 1 and 8.

[77] D. CAMPAGNA, C. KAVKA, A. TURCO, B. POGACE, AND C. POLONI, *Solving time-dependent coupled systems through FMI co-simulation and BPMN process orchestration*, in IEEE International Symposium on Systems Engineering (ISSE), Edinburgh, Scotland, Oct. 2016, IEEE, pp. 1–8. Cited on page 177.

[78] B. CAMUS, C. BOURJOT, AND V. CHEVRIER, *Combining DEVS with multi-agent concepts to design and simulate multi-models of complex systems (WIP)*, in Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium, Society for Computer Simulation International, 2015, pp. 85–90. Cited on page 61.

[79] B. CAMUS, V. GALTIER, M. CAUJOLLE, V. CHEVRIER, J. VAUBOURG, L. CIARLETTA, AND C. BOURJOT, *Hybrid Co-simulation of FMUs using DEV&DESS in MECSYCO*, in Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium (TMS/DEVS 16), Pasadena, CA, United States, 2016, Society for Computer Simulation International San Diego, CA, USA, p. No. 8. Cited on pages 61, 62, and 146.

[80] L. P. CARLONI, R. PASSERONE, A. PINTO, AND A. L. ANGIOVANNI-VINCENTELLI, *Languages and Tools for Hybrid Systems Design*, Foundations and Trends® in Electronic Design Automation, 1 (2006), pp. 1–193. Cited on page 60.

[81] C. D. CAROTHERS, K. S. PERUMALLA, AND R. M. FUJIMOTO, *Efficient optimistic parallel simulations using reverse computation*, ACM Transactions on Modeling and Computer Simulation, 9 (1999), pp. 224–253. Place: New York, NY, USA. Cited on page 18.

[82] A. B. CARROLL AND R. T. WETHERALD, *Application of Parallel Processing to Numerical Weather Prediction*, Journal of the ACM, 14 (1967), pp. 591–614. Cited on page 16.

[83] R. CARTER AND E. M. NAVARRO-LÓPEZ, *Dynamically-Driven Timed Automaton Abstractions for Proving Liveness of Continuous Systems*, in Formal Modeling and Analysis of Timed Systems, London, United Kingdom, 2012, Springer Berlin Heidelberg, pp. 59–74. Cited on page 22.

[84] F. E. CELLIER, *Combined Continuous/Discrete System Simulation Languages: Usefulness, Experiences and Future Development*, Special Interest Group (SIG) on SImulation and Modeling (SIM), 9 (1977), pp. 18–21. Place: New York, NY, USA. Cited on page 60.

[85] ——, *Combined Continuous Discrete Simulation by Use of Digital Computers: Techniques and Tools*, PhD thesis, Swiss Federal Institute of Technology Zurich, 1979. Cited on page 58.

[86] ——, *Continuous System Modeling*, Springer Science & Business Media, 1991. Cited on page 6.

[87] F. E. CELLIER AND E. KOFMAN, *Continuous System Simulation*, Springer Science & Business Media, 2006. Cited on pages 2, 11, 39, 52, 58, 63, 89, 90, 115, 133, 144, and 145.

[88] S. CENTOMO, J. DEANTONI, AND R. DE SIMONE, *Using SystemC Cyber Models in an FMI Co-Simulation Environment: Results and Proposed FMI Enhancements*,

in Euromicro Conference on Digital System Design (DSD), Limassol, Cyprus, Aug. 2016, IEEE, pp. 318–325. Cited on page 176.

[89] K. CHANDY AND J. MISRA, *Distributed Simulation: A Case Study in Design and Verification of Distributed Programs*, IEEE Transactions on Software Engineering, SE-5 (1979), pp. 440–452. Cited on page 36.

[90] W.-T. CHANG, A. KALAVADE, AND E. A. LEE, *Effective Heterogenous Design and Co-Simulation*, in Hardware/Software Co-Design, Springer Netherlands, Dordrecht, 1996, pp. 187–212. Cited on page 18.

[91] D. CHAZAN AND W. L. MIRANKER, *A Nongradient and Parallel Algorithm for Unconstrained Minimization*, SIAM Journal on Control, 8 (1970), pp. 207–217. Cited on page 16.

[92] B.-C. CHEN AND H. PENG, *Differential-Braking-Based Rollover Prevention for Sport Utility Vehicles with Human-in-the-loop Evaluations*, Vehicle System Dynamics, 36 (2001), pp. 359–389. Cited on page 2.

[93] K. B. CLARK, *Project scope and project performance: The effect of parts strategy and supplier involvement on product development*, Management science, 35 (1989), pp. 1247–1263. Cited on pages 1 and 8.

[94] E. M. CLARKE, O. GRUMBERG, AND D. E. LONG, *Model checking and abstraction*, ACM Transactions on Programming Languages and Systems, 16 (1994), pp. 1512–1542. Cited on page 7.

[95] M. J. CLAYTON, *Delphi: A technique to harness expert opinion for critical decision-making tasks in education*, Educational Psychology, 17 (1997), pp. 373–386. Cited on pages 74 and 75.

[96] F. CONTI AND O. KHATIB, *A Framework for Real-Time Multi-Contact Multi-Body Dynamic Simulation*, in Robotics Research, vol. 114, Springer International Publishing, Cham, 2016, pp. 271–287. Cited on page 135.

[97] {CONTROLLAB PRODUCTS}, *Design of a Compensated Motion Crane using INTO-CPS*, tech. rep., Press Release EU, Enschede, Netherlands, 2018. Cited on pages vii, 24, and 25.

[98] 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, Nov. 2016, IEEE. Cited on page 176.

[99] F. CREMONA, M. LOHSTROH, D. BROMAN, E. A. LEE, M. MASIN, AND S. TRIPAKIS, *Hybrid co-simulation: It's about time*, Software & Systems Modeling, 10270 (2017). Cited on pages 64 and 83.

[100] 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, 2016, ACM New York, NY, USA, pp. 1759–1766. Cited on pages 61 and 176.

[101] X. DAI, *A Gel'fand-type spectral radius formula and stability of linear constrained switching systems*, Linear Algebra and its Applications, 436 (2012), pp. 1099–1113. Cited on pages 101, 102, and 121.

[102] N. DALKEY AND O. HELMER, *An Experimental Application of the Delphi Method to the Use of Experts*, Management Science, 9 (1963), pp. 458–467. Cited on page 74.

[103] J. DAVIS II, M. GOEL, C. HYLANDS, B. KIENHUIS, E. A. LEE, J. LIU, X. LIU, L. MULIADI, S. NEUENDORFFER, AND J. REEKIE, *Overview of the Ptolemy project*, tech. rep., ERL Technical Report UCB/ERL, 1999. Cited on page 176.

[104] C. W. DE SILVA, *Mechatronics: An Integrated Approach*, CRC press, 2004. Cited on pages 1 and 8.

[105] J. DEANTONI, *Modeling the Behavioral Semantics of Heterogeneous Languages and their Coordination*, in Architecture-Centric Virtual Integration (ACVI), Venice, Italy, Apr. 2016, IEEE, pp. 12–18. Cited on page 176.

[106] A. L. DELBECQ, A. H. VAN DE VEN, AND D. H. GUSTAFSON, *Group Techniques for Program Planning: A Guide to Nominal Group and Delphi Processes*, Scott-Foresman and Company, Glenview, Illinois, 1975. Cited on page 75.

[107] J. DENIL, *Design, Verification and Deployment of Software Intensive Systems - A Multiparadigm Approach*, PhD thesis, University of Antwerp, 2013. Cited on pages 7 and 147.

[108] J. DENIL, P. DE MEULENAERE, S. DEMEYER, AND H. VANGHELUWE, *DEVS for AUTOSAR-based system deployment modeling and simulation*, SIMULATION, 93 (2017), pp. 489–513. Cited on page 2.

[109] J. DENIL, S. KLIKOVITS, P. J. MOSTERMAN, A. VALLECILLO, AND H. VANGHELUWE, *The experiment model and validity frame in M&S*, in Proceedings of the Symposium on Theory of Modeling & Simulation, vol. 49, Virginia Beach, Virginia, USA, 2017, Society for Computer Simulation International, p. Article No. 10. Cited on pages 6, 79, 83, and 196.

[110] J. DENIL, B. MEYERS, P. DE MEULENAERE, AND H. VANGHELUWE, *Explicit Semantic Adaptation of Hybrid Formalisms for FMI Co-Simulation*, in Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium, Alexandria, Virginia, Apr. 2015, Society for Computer Simulation International San Diego, CA, USA, pp. 99–106. Cited on pages 61, 62, 137, 138, 145, and 151.

[111] J. DINGLIANA AND C. O'SULLIVAN, *Graceful Degradation of Collision Handling in Physically Based Animation*, Computer Graphics Forum, 19 (2000), pp. 239–248. Cited on page 135.

[112] P. A. M. DIRAC, *The Principles of Quantum Mechanics*, Oxford university press, 1981. Cited on page 62.

[113] M. C. F. DONKERS, W. P. M. H. HEEMELS, N. VAN DE WOUW, AND L. HETEL, *Stability Analysis of Networked Control Systems Using a Switched Linear Systems Approach*, IEEE Transactions on Automatic Control, 56 (2011), pp. 2101–2115. Cited on pages 117 and 135.

[114] E. DRAUGHON, R. GRISHMAN, J. SCHWARTZ, AND A. STEIN, *Programming considerations for parallel computers*, Tech. Rep. IMM 362, Courant Institute, New York, NY, USA, 1967. Cited on page 16.

[115] E. DRENTH, M. TÖRMÄNEN, K. JOHANSSON, B.-A. ANDERSSON, D. ANDERSSON, I. TORSTENSSON, AND J. ÅKESSON, *Consistent Simulation Environment with FMI based Tool Chain*, in 10th International Modelica Conference, 2014. Cited on page 19.

[116] S. DRONKA AND J. RAUH, *Co-simulation-interface for user-force-elements*, in SIMPACK User Meeting, Baden-Baden, Germany, Mar. 2006. Cited on pages 43, 58, 146, 174, and 194.

[117] J. EKER, J. JANNECK, E. LEE, JIE LIU, XIAOJUN LIU, J. LUDVIG, S. NEUENDORFFER, S. SACHS, AND YUHONG XIONG, *Taming heterogeneity - the Ptolemy approach*, Proceedings of the IEEE, 91 (2003), pp. 127–144. Cited on page 19.

[118] H. EL TAHAWY, D. RODRIGUEZ, S. GARCIA-SABIRO, AND J.-J. MAYOL, *VHD/-sub e/LDO: A new mixed mode simulation*, in Proceedings of EURO-DAC 93 and EURO-VHDL 93- European Design Automation Conference, IEEE Comput. Soc. Press, 1993, pp. 546–551. Cited on page 18.

[119] L. ELSNER, *The generalized spectral-radius theorem: An analytic-geometric proof*, Linear Algebra and its Applications, 220 (1995), pp. 151–159. Cited on page 116.

[120] O. ENGE-ROSENBLATT, C. CLAUSS, A. SCHNEIDER, AND P. SCHNEIDER, *Functional Digital Mock-up and the Functional Mock-up Interface–Two Complementary Approaches for a Comprehensive Investigation of Heterogeneous Systems*, in 8th International Modelica Conference, Dresden, Germany, 2011, Linköping University Electronic Press; Linköpings universitet, pp. 748–755. Cited on pages 45 and 174.

[121] K. ERIKSSON, D. ESTEP, AND C. JOHNSON, *Applied Mathematics: Body and Soul*, Springer Berlin Heidelberg, Berlin, Heidelberg, volume 1 ed., 2004. Cited on page 53.

[122] Y. A. FELDMAN, L. GREENBERG, AND E. PALACHI, *Simulating Rhapsody SysML Blocks in Hybrid Models with FMI*, in 10th International Modelica Conference, Lund, Sweden, Mar. 2014, Linköping University Electronic Press, pp. 43–52. Cited on pages 61 and 176.

[123] C. A. FELIPPA AND T. L. GEERS, *Partitioned analysis for coupled mechanical systems*, Engineering Computations, 5 (1988), pp. 123–133. Cited on page 16.

[124] C. A. FELIPPA, K. PARK, AND C. FARHAT, *Partitioned analysis of coupled mechanical systems*, Computer Methods in Applied Mechanics and Engineering, 190 (2001), pp. 3247–3270. Cited on pages 16 and 18.

[125] P. FEY, H. CARTER, AND P. WILSEY, *Parallel synchronization of continuous time discrete event simulators*, in International Conference on Parallel Processing (Cat. No.97TB100162), IEEE Comput. Soc, 1997, pp. 227–231. Cited on pages 18 and 61.

[126] FMI, *Functional Mock-up Interface for Model Exchange and Co-Simulation*, tech. rep., FMI development group, 2014. Cited on pages 66 and 67.

[127] F. F. FOLDAGER, P. G. LARSEN, AND O. GREEN, *Development of a Driverless Lawn Mower Using Co-simulation*, in Software Engineering and Formal Methods, vol. 10729, Springer International Publishing, Cham, 2018, pp. 330–344. Cited on pages vii, 23, and 25.

[128] J. J. FONTANELLA, *The Physics of Basketball*, JHU Press, 2008. Cited on page 117.

[129] G. FREHSE, C. LE GUERNIC, A. DONZÉ, S. COTTON, R. RAY, O. LEBELTEL, R. RIPADO, A. GIRARD, T. DANG, AND O. MALER, *SpaceEx: Scalable Verification of Hybrid Systems*, in Computer Aided Verification, Springer Berlin Heidelberg, 2011, pp. 379–395. Cited on page 134.

[130] P. FREY, R. RADHAKRISHNAN, H. W. CARTER, AND P. A. WILSEY, *Optimistic synchronization of mixed-mode simulators*, in 1998 First Merged International on Parallel Processing Symposium and Symposium on Parallel and Distributed Processing, 1998, pp. 694–699. Cited on page 18.

[131] J. FRIEDMAN AND J. GHIDELLA, *Using Model-Based Design for Automotive Systems Engineering - Requirements Analysis of the Power Window Example*, in Transactions Journal of Passenger Cars: Electronic and Electrical Systems, vol. 115 of Automotive Systems Engineering, Detroit, USA, Apr. 2006, SAE Technical Paper, p. 8. Cited on page 1.

[132] M. FRIEDRICH, *Parallel Co-Simulation for Mechatronic Systems*, PhD thesis, Fakultät für Maschinenwesen, Germany, 2011. Cited on pages 45 and 174.

[133] P. FRITZSON, P. ARONSSON, A. POP, H. LUNDVALL, K. NYSTROM, L. SALDAMLI, D. BROMAN, AND A. SANDHOLM, *OpenModelica - A free open-source environment for system modeling, simulation, and teaching*, in Conference on Computer Aided Control System Design, International Conference on Control Applications, International Symposium on Intelligent Control, Munich, Germany, Oct. 2006, IEEE, pp. 1588–1595. Cited on page 148.

[134] R. M. FUJIMOTO, *Parallel discrete event simulation*, Communications of the ACM, 33 (1990), pp. 30–53. Place: New York, NY, USA. Cited on page 16.

[135] ——, *Parallel and distributed simulation systems*, in Winter Simulation Conference (Cat. No.01CH37304), vol. 300, Arlington, VA, USA, 2001, Wiley New York, pp. 147–157. Cited on pages 6, 36, 37, and 63.

[136] V. GALTIER, S. VIALLE, C. DAD, J.-P. TAVELLA, J.-P. LAM-YEE-MUI, AND G. PLESSIS, *FMI-Based Distributed Multi-Simulation with DACCOSIM*, in Spring Simulation Multi-Conference, Alexandria, Virginia, USA, 2015, Society for Computer Simulation International San Diego, CA, USA, pp. 804–811. Cited on pages 19, 45, 52, 53, 62, 150, 174, 177, and 194.

[137] C. E. GARCÍA, D. M. PRETT, AND M. MORARI, *Model predictive control: Theory and practice—A survey*, Automatica, 25 (1989), pp. 335–348. Cited on pages 17 and 88.

[138] D. GARLAN AND MARY SHAW, *An Introduction to Software Architecture*, Advances in Software Engineering and Knowledge Engineering, I (1993). Cited on page 17.

[139] A. GARRO AND A. FALCONE, *On the integration of HLA and FMI for supporting interoperability and reusability in distributed simulation*, in Spring Simulation Multi-Conference, Society for Computer Simulation International, 2015, pp. 774–781. Cited on pages 61 and 64.

[140] C. W. GEAR AND D. R. WELLS, *Multirate linear multistep methods*, BIT, 24 (1984), pp. 484–502. Cited on page 17.

[141] C. W. GEART AND D. S. WATANABE, *Stability and Convergence of Variable Order Multistep Methods*, SIAM Journal on Numerical Analysis, 11 (1974), pp. 1044–1058. Cited on page 89.

[142] D. GELERNTER AND N. CARRIERO, *Coordination Languages and Their Significance*, Commun. ACM, 35 (1992), pp. 96–. Cited on page 17.

[143] L. GHEORGHE, F. BOUCHHIMA, G. NICOLESCU, AND H. BOUCHENEB, *A Formalization of Global Simulation Models for Continuous/Discrete Systems*, in Summer Computer Simulation Conference, San Diego, CA, USA, July 2007, Society for Computer Simulation International San Diego, CA, USA, pp. 559–566. Series Title: SCSC '07. Cited on pages 143 and 177.

[144] A. GHOSH, M. BERSHTEYN, R. CASLEY, C. CHIEN, A. JAIN, M. LIPSIE, D. TARRODAYCHIK, AND O. YAMAMOTO, *A hardware-software co-simulator for embedded system design and debugging*, in Design Automation Conference, Chiba, Japan, 1995, pp. 155–164. Cited on page 63.

[145] F. GIANNAKOPOULOS AND K. PLIETE, *Planar systems of piecewise linear differential equations with a line of discontinuity*, Nonlinearity, 14 (2001), pp. 1611–1632. Cited on pages 135 and 136.

[146] A. GIRARD, G. POLA, AND P. TABUADA, *Approximately bisimilar symbolic models for incrementally stable switched systems*, IEEE Transactions on Automatic Control, 55 (2010), pp. 116–126. Cited on page 132.

[147] E. GLAESSGEN AND D. STARGEL, *The Digital Twin Paradigm for Future NASA and U.S. Air Force Vehicles*, in Structures, Structural Dynamics, and Materials Conference: Special Session on the Digital Twin, Reston, Virigina, Apr. 2012, American Institute of Aeronautics and Astronautics, pp. 1–14. Cited on pages 19 and 20.

[148] R. GOEBEL, R. G. SANFELICE, AND A. R. TEEL, *Hybrid dynamical systems*, IEEE Control Systems, 29 (2009), pp. 28–93. Cited on page 120.

[149] ——, *Hybrid Dynamical Systems: Modeling, Stability, and Robustness*, Princeton University Press, 2012. Publication Title: Book. Cited on page 116.

[150] C. GOMES, *Foundations for Continuous Time Hierarchical Co-simulation*, in ACM Student Research Competition (MoDELS), Saint Malo, France, Oct. 2016, ACM New York, NY, USA, p. to appear. Cited on pages 47, 64, 144, and 145.

[151] C. GOMES, J. DENIL, AND H. VANGHELUWE, *Causal-Block Diagrams*, tech. rep., University of Antwerp, 2016. Cited on page 141.

[152] C. GOMES, R. JUNGERS, B. LEGAT, AND H. VANGHELUWE, *Minimally Constrained Stable Switched Systems and Application to Co-simulation*, Tech. Rep. arXiv:1809.02648, University of Antwerp, Belgium, 2018. Cited on page 121.

[153] C. GOMES, P. KARALIS, E. M. NAVARRO-LÓPEZ, AND H. VANGHELUWE, *Approximated Stability Analysis of Bi-modal Hybrid Co-simulation Scenarios*, in 1st Workshop on Formal Co-Simulation of Cyber-Physical Systems, Trento, Italy, 2017, Springer, Cham, pp. 345–360. Cited on pages 63, 83, 113, and 138.

[154] C. GOMES, B. LEGAT, R. JUNGERS, AND H. VANGHELUWE, *Minimally Constrained Stable Switched Systems and Application to Co-simulation*, in IEEE Conference on Decision and Control, Miami Beach, FL, USA, 2018, p. to be published. Cited on pages 83 and 121.

[155] C. GOMES, B. LEGAT, R. M. JUNGERS, AND H. VANGHELUWE, *Stable Adaptive Co-simulation: A Switched Systems Approach*, in IUTAM Symposium on Co-Simulation and Solver Coupling, Darmstadt, Germany, 2017, p. to appear. Cited on pages 63, 83, 103, and 138.

[156] C. GOMES, B. MEYERS, J. DENIL, C. THULE, K. LAUSDAHL, H. VANGHELUWE, AND P. DE MEULENAERE, *Semantic Adaptation for FMI Co-simulation with Hierarchical Simulators*, SIMULATION, 95 (2018), pp. 1–29. Cited on pages 62, 64, and 83.

[157] C. GOMES, C. THULE, D. BROMAN, P. G. LARSEN, AND H. VANGHELUWE, *Co-simulation: State of the art*, tech. rep., University of Antwerp, Feb. 2017. Cited on pages 9, 22, 23, 83, 138, 140, 144, 145, and 174.

[158] ——, *Co-simulation: A Survey*, ACM Computing Surveys, 51 (2018), p. Article 49. Cited on pages 9, 83, 87, and 186.

[159] C. GOMES, C. THULE, P. G. LARSEN, J. DENIL, AND H. VANGHELUWE, *Co-simulation of Continuous Systems: A Tutorial*, Tech. Rep. arXiv:1809.08463 [cs, math], University of Antwerp, Sept. 2018. Cited on page 179.

[160] C. GOMES, Y. VAN TENDELOO, J. DENIL, P. DE MEULENAERE, AND H. VANGHELUWE, *Hybrid System Modelling and Simulation with Dirac Deltas*, in Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium, Virginia Beach, Virginia, USA, 2017, Society for Computer Simulation International, p. Article No. 7. Series Title: DEVS '17. Cited on page 62.

[161] F. GONZÁLEZ, S. ARBATANI, A. MOHTAT, AND J. KÖVECSES, *Energy-leak monitoring and correction to enhance stability in the co-simulation of mechanical systems*, Mechanism and Machine Theory, 131 (2019), pp. 172–188. Cited on pages 54, 71, 184, and 194.

[162] F. GONZÁLEZ, M. A. NAYA, A. LUACES, AND M. GONZÁLEZ, *On the effect of multirate co-simulation techniques in the efficiency and accuracy of multibody system dynamics*, Multibody System Dynamics, 25 (2011), pp. 461–483. Cited on page 145.

[163] M. GRIEVES AND J. VICKERS, *Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems*, in Transdisciplinary Perspectives on Complex Systems, Springer International Publishing, Cham, 2017, pp. 85–113. Cited on page 20.

[164] G. GRIPENBERG, *Computing the joint spectral radius*, Linear Algebra and its Applications, 234 (1996), pp. 43–60. Cited on pages 97 and 102.

[165] B. GU AND H. H. ASADA, *Co-simulation of algebraically coupled dynamic subsystems*, in American Control Conference, vol. 3, Arlington, VA, USA, 2001, IEEE, pp. 2273–2278. Cited on pages 19, 45, 47, 57, 97, and 174.

[166] ——, *Co-Simulation of Algebraically Coupled Dynamic Subsystems Without Disclosure of Proprietary Subsystem Models*, Journal of Dynamic Systems, Measurement, and Control, 126 (2004), p. 1. Cited on pages 47 and 174.

[167] N. GUGLIELMI AND M. ZENNARO, *An algorithm for finding extremal polytope norms of matrix families*, Linear Algebra and its Applications, 428 (2008), pp. 2265–2282. Cited on pages 97 and 102.

[168] GUISHENG ZHAI, BO HU, K. YASUDA, AND A. MICHEL, *Qualitative analysis of discrete-time switched systems*, in Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301), IEEE, 2002, pp. 1880–1885 vol.3. Cited on page 113.

[169] R. K. GUPTA, C. N. COELHO JR., AND G. DE MICHELI, *Synthesis and Simulation of Digital Systems Containing Interacting Hardware and Software Components*, in Proceedings of the 29th ACM/IEEE Design Automation Conference, Los Alamitos, CA, USA, 1992, IEEE Computer Society Press, pp. 225–230. Series Title: DAC '92. Cited on page 18.

[170] I. HAFNER, B. HEINZL, AND M. ROESSLER, *An Investigation on Loose Coupling Co-Simulation with the BCVTB*, SNE Simulation Notes Europe, 23 (2013). Cited on pages 53 and 83.

[171] I. HAFNER AND N. POPPER, *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, New York, New York, USA, 2017, ACM Press, pp. 67–76. Cited on page 26.

[172] M. R. HALLOWELL AND J. A. GAMBATESE, *Qualitative Research: Application of the Delphi Method to CEM Research*, Journal of Construction Engineering and Management, 136 (2010), pp. 99–107. Cited on page 75.

[173] F. HASSON, S. KEENEY, AND H. MCKENNA, *Research guidelines for the Delphi survey technique*, Journal of Advanced Nursing, (2000). Cited on page 76.

[174] W. P. M. H. HEEMELS, A. R. TEEL, N. VAN DE WOUW, AND D. NEŠIĆ, *Networked Control Systems With Communication Constraints: Tradeoffs Between Transmission Intervals, Delays and Performance*, IEEE Transactions on Automatic Control, 55 (2010), pp. 1781–1796. Cited on page 135.

[175] W. P. M. H. HEEMELS, N. VAN DE WOUW, R. H. GIELEN, M. C. F. DONKERS, L. HETEL, S. OLARU, M. LAZAR, J. DAAFOUZ, AND S. NICULESCU, *Comparison*

of overapproximation methods for stability analysis of networked control systems, in 13th ACM International Conference on Hybrid Systems: Computation and Control - HSCC '10, Stockholm, Sweden, 2010, ACM Press, p. 181. Cited on page 135.

[176] E. HERNANDEZ-VARGAS, P. COLANERI, R. MIDDLETON, AND F. BLANCHINI, *Discrete-time control for switched positive systems with application to mitigating viral escape*, International Journal of Robust and Nonlinear Control, 21 (2011), pp. 1093–1111. Cited on page 96.

[177] J. HESPANHA, P. NAGHSHTABRIZI, AND Y. XU, *A Survey of Recent Results in Networked Control Systems*, Proceedings of the IEEE, 95 (2007), pp. 138–162. Cited on page 135.

[178] L. HETEL, J. DAAFOUZ, AND C. IUNG, *Stabilization of Arbitrary Switched Linear Systems With Unknown Time-Varying Delays*, IEEE Transactions on Automatic Control, 51 (2006), pp. 1668–1674. Cited on page 113.

[179] D. R. HICKEY, P. A. WILSEY, R. J. HOEKSTRA, E. R. KEITER, S. A. HUTCHINSON, AND T. V. RUSSO, *Mixed-signal simulation with the Simbus backplane*, in 39th Annual Simulation Symposium, 2006, Huntsville, AL, 2006. Cited on page 18.

[180] A. HIMMLER, *Hardware-in-the-Loop Technology Enabling Flexible Testing Processes*, in 51st AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition, Grapevine (Dallas/Ft. Worth Region), Texas, Jan. 2013, American Institute of Aeronautics and Astronautics, pp. 1–8. Cited on page 2.

[181] K. HINES AND G. BORRIELLO, *Dynamic Communication Models in Embedded System Co-simulation*, in Proceedings of the 34th Annual Design Automation Conference, New York, NY, USA, 1997, ACM, pp. 395–400. Series Title: DAC '97. Cited on page 18.

[182] ——, *Selective focus as a means of improving geographically distributed embedded system co-simulation*, in 8th IEEE International Workshop on Rapid System Prototyping, 1997, 1997, pp. 58–62. Cited on pages 18 and 88.

[183] M. W. HIRSCH, S. SMALE, AND R. L. DEVANEY, *Differential Equations, Dynamical Systems, and an Introduction to Chaos*, Academic press, 2012. Cited on pages 122 and 123.

[184] M. HOEPFER, *Towards a Comprehensive Framework for Co- Simulation of Dynamic Models With an Emphasis on Time Stepping*, PhD thesis, Georgia Institute of Technology, USA, 2011. Cited on pages 51, 54, 174, and 194.

[185] F. R. HOLZINGER AND M. BENEDIKT, *Hierarchical Coupling Approach Utilizing Multi-Objective Optimization for Non-Iterative Co-Simulation*, in 13th International Modelica Conference 2019, Regensburg, Germany, 2019, Linköping University Electronic Press, p. 6. Cited on page 195.

[186] C.-C. HSU AND B. SANDFORD, *The delphi technique: Making sense of consensus*, Practical Assessment, Research & Evaluation, 12 (2007), pp. 1–8. Cited on page 74.

[187] T. J. R. HUGHES AND W. K. LIU, *Implicit-Explicit Finite Elements in Transient Analysis: Implementation and Numerical Examples*, Journal of Applied Mechanics, 45 (1978), p. 375. Cited on page 16.

[188] IEEE, *IEEE Standard for Distributed Interactive Simulation–Application Protocols*, Tech. Rep. 10.1109/IEEESTD.2012.6387564, IEEE, 2012. Cited on page 38.

[189] F. IMMLER, *Formally Verified Computation of Enclosures of Solutions of Ordinary Differential Equations*, in NASA Formal Methods, vol. 8430 LNCS, Houston, TX, USA, 2014, Springer Berlin Heidelberg, pp. 113–127. Cited on page 22.

[190] K. R. JACKSON, *A survey of parallel numerical methods for initial value problems for ordinary differential equations*, IEEE Transactions on Magnetics, 27 (1991), pp. 3792–3797. Cited on page 16.

[191] A. JANTSCH AND I. SANDER, *Models of computation and languages for embedded system design*, IEE Proceedings - Computers and Digital Techniques, 152 (2005), pp. 114–129(15). Cited on pages 6 and 9.

[192] D. R. JEFFERSON, *Virtual time*, ACM Transactions on Programming Languages and Systems, 7 (1985), pp. 404–425. Place: New York, NY, USA. Cited on page 36.

[193] H. H. JO, H. R. PARSAEI, AND W. G. SULLIVAN, *Principles of concurrent engineering*, in Concurrent Engineering, Springer US, Boston, MA, 1993, pp. 3–23. Cited on pages 1 and 17.

[194] K. H. JOHANSSON, M. EGERSTEDT, J. LYGEROS, AND S. SASTRY, *On the regularization of Zeno hybrid automata*, Systems & Control Letters, 38 (1999), pp. 141–150. Cited on pages 63 and 118.

[195] M. JOHANSSON, *Piecewise Linear Control Systems*, vol. 284, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. Series Title: Lecture Notes in Control and Information Sciences. Cited on page 116.

[196] N. JØRGENSEN, *The Boeing 777: Development Life Cycle Follows Artifact*, in World Conference on Integrated Design and Process Technology (IDPT), Citeseer, 2006, pp. 25–30. Cited on pages 1 and 17.

[197] R. JUNGERS, *The Joint Spectral Radius: Theory and Applications*, vol. 385, Springer Science & Business Media, 2009. Cited on pages 63, 88, 96, 97, 98, 101, 102, 110, 113, and 116.

[198] R. M. JUNGERS, A. CICONE, AND N. GUGLIELMI, *Lifted Polytope Methods for Computing the Joint Spectral Radius*, SIAM Journal on Matrix Analysis and Applications, 35 (2014), pp. 391–410. Cited on pages 97 and 102.

[199] R. M. JUNGERS, A. D'INNOCENZO, AND M. D. DI BENEDETTO, *Feedback stabilization of dynamical systems with switched delays*, in 2012 IEEE 51st IEEE Conference on Decision and Control (CDC), Maui, HI, USA, Dec. 2012, IEEE, pp. 1325–1330. Cited on page 135.

[200] R. M. JUNGERS, V. PROTASOV, AND V. D. BLONDEL, *Efficient algorithms for deciding the type of growth of products of integer matrices*, Linear Algebra and its Applications, 428 (2008), pp. 2296–2311. Cited on page 96.

[201] T. KALMAR-NAGY AND I. STANCIULESCU, *Can complex systems really be simulated?*, Applied Mathematics and Computation, 227 (2014), pp. 199–211. Cited on pages 51, 57, and 91.

[202] K. C. KANG, S. COHEN, J. HESS, W. NOVAK, AND A. PETERSON, *Feature-Oriented Domain Analysis. Feasibility study,*, tech. rep., Carnegie Mellon University, 1990. Cited on pages 65 and 190.

[203] P. KARALIS AND E. M. NAVARRO-LÓPEZ, *Feedback stability for dissipative switched systems*, in Proceedings of the 20th IFAC World Congress, Toulouse, France, 2017, IFAC, pp. 3442–3448. Cited on page 116.

[204] M. KARNER, E. ARMENGAUD, C. STEGER, AND R. WEISS, *Holistic Simulation of FlexRay Networks by Using Run-time Model Switching*, in Proceedings of the Conference on Design, Automation and Test in Europe, 3001 Leuven, Belgium, Belgium, 2010, European Design and Automation Association, pp. 544–549. Series Title: DATE '10. Cited on pages 19 and 88.

[205] S. KELLY AND J.-P. TOLVANEN, *Domain-Specific Modeling: Enabling Full Code Generation*, John Wiley & Sons, 2008. Cited on pages 147 and 172.

[206] J. KENT PEACOCK, J. WONG, AND E. G. MANNING, *Distributed simulation using a network of processors*, Computer Networks (1976), 3 (1979), pp. 44–56. Cited on page 16.

[207] H. K. KHALIL, *Nonlinear Systems*, Prentice-Hall, New Jersey, 1996. Cited on page 116.

[208] R. B. KIEBURTZ, L. MCKINNEY, J. M. BELL, J. HOOK, A. KOTOV, J. LEWIS, D. P. OLIVA, T. SHEARD, I. SMITH, AND L. WALTON, *A software engineering experiment in software component generation*, in 18th International Conference on Software Engineering, Berlin, Germany, Mar. 1996, IEEE Computer Society, pp. 542–552. Cited on page 172.

[209] J. E. KLECKNER, *Advanced Mixed-Mode Simulation Techniques*, ph.D., University of California, Berkeley, 1984. Cited on page 18.

[210] E. KOFMAN, *A Second-Order Approximation for DEVS Simulation of Continuous Systems*, SIMULATION, 78 (2002), pp. 76–89. Cited on page 62.

[211] E. KOFMAN AND S. JUNCO, *Quantized-state systems: A DEVS Approach for continuous system simulation*, Transactions of The Society for Modeling and Simulation International, 18 (2001), pp. 123–132. Cited on pages 61, 62, and 146.

[212] A. KOSSIAKOFF, W. N. SWEET, S. J. SEYMOUR, AND S. M. BIEMER, *Structure of Complex Systems*, in Systems Engineering Principles and Practice, John Wiley & Sons, Inc., Hoboken, NJ, USA, Mar. 2011, pp. 41–67. Cited on page 36.

[213] V. KOUNEV, D. TIPPER, M. LEVESQUE, B. M. GRAINGER, T. MCDERMOTT, AND G. F. REED, *A microgrid co-simulation framework*, in Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), Lévesque, McDermott, Apr. 2015, IEEE, pp. 1–6. Cited on page 61.

[214] M. KRAMMER, J. FRITZ, AND M. KARNER, *Model-Based Configuration of Automotive Co-Simulation Scenarios*, in 48th Annual Simulation Symposium, Alexandria, Virginia, 2015, Society for Computer Simulation International San Diego, CA, USA, pp. 155–162. Cited on pages 45, 174, and 195.

[215] P. KRUS, *Modeling of Mechanical Systems Using Rigid Bodies and Transmission Line Joints*, Journal of Dynamic Systems, Measurement, and Control, 121 (1999), p. 606. Cited on page 184.

[216] R. KÜBLER AND W. SCHIEHLEN, *Modular Simulation in Multibody System Dynamics*, Multibody System Dynamics, 4 (2000), pp. 107–127. Cited on pages 2, 57, and 91.

[217] ——, *Two Methods of Simulator Coupling*, Mathematical and Computer Modelling of Dynamical Systems, 6 (2000), pp. 93–113. Cited on pages 19, 50, 51, 53, 83, 140, and 141.

[218] F. KUHL, R. WEATHERLY, AND J. DAHMANN, *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*, Prentice Hall PTR, 1999. Cited on page 18.

[219] T. KÜHNE, *What is a Model?*, in Language Engineering for Model-Driven Software Development, vol. 04101, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), 2005. Cited on page 6.

[220] T. KUHR, T. FORSTER, T. BRAUN, AND R. GOTZHEIN, *FERAL - Framework for simulator coupling on requirements and architecture level*, in Eleventh IEEE/ACM International Conference on Formal Methods and Models for Codesign (MEMOCODE), Portland, OR, USA, 2013, IEEE, pp. 11–22. Cited on pages 61, 62, and 69.

[221] A. KUNDU AND D. CHATTERJEE, *Stabilizing switching signals: A transition from point-wise to asymptotic conditions*, Systems & Control Letters, 106 (2017), pp. 16–23. Cited on page 113.

[222] S. LACOSTE-JULIEN, H. VANGHELUWE, J. DE LARA, AND P. J. MOSTERMAN, *Meta-modelling hybrid formalisms*, in IEEE International Symposium on Computer Aided Control Systems Design, New Orleans, LA, USA, Sept. 2004, IEEE, pp. 65–70. Cited on page 176.

[223] J. C. LAGARIAS AND Y. WANG, *The finiteness conjecture for the generalized spectral radius of a set of matrices*, Linear Algebra and its Applications, 214 (1995), pp. 17–42. Cited on page 121.

[224] L. LAMPORT, *Time, clocks, and the ordering of events in a distributed system*, Communications of the ACM, 21 (1978), pp. 558–565. Place: New York, NY, USA. Cited on pages 16 and 36.

[225] J. LANDETA, *Current validity of the Delphi method in social sciences*, Technological Forecasting and Social Change, 73 (2006), pp. 467–482. Cited on page 74.

[226] P. G. LARSEN, J. FITZGERALD, J. WOODCOCK, P. FRITZSON, J. BRAUER, C. KLEIJN, T. LECOMTE, M. PFEIL, O. GREEN, S. BASAGIANNIS, AND A. SADOVYKH, *Integrated tool chain for model-based design of Cyber-Physical Systems: The INTO-CPS project*, in 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data), Vienna, Austria, Apr. 2016, IEEE, pp. 1–6. Cited on pages 150 and 177.

[227] D. P. Y. LAWRENCE, C. GOMES, J. DENIL, H. VANGHELUWE, AND D. BUCHS, *Coupling Petri nets with Deterministic Formalisms Using Co-simulation*, in Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium, Pasadena, CA, USA, 2016, pp. 6:1–6:8. Cited on pages 10, 61, and 83.

[228] P. LE MARREC, C. A. VALDERRAMA, F. HESSEL, A. A. JERRAYA, M. ATTIA, AND O. CAYROL, *Hardware, software and mechanical cosimulation for automotive applications*, in 9th International Workshop on Rapid System Prototyping, 1998, pp. 202–206. Cited on page 18.

[229] B.-S. LEE, W. CAI, S. J. TURNER, AND L. CHEN, *Adaptive dead reckoning algorithms for distributed interactive simulation*, International Journal of Simulation, 1 (2000), pp. 21–34. Cited on page 37.

[230] E. A. LEE, *Cyber Physical Systems: Design Challenges*, in 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC), 2008, pp. 363–369. Cited on pages 1 and 8.

[231] E. A. LEE AND A. SANGIOVANNI-VINCENTELLI, *A framework for comparing models of computation*, Transactions on Computer-Aided Design of Integrated Circuits and Systems, 17 (1998), pp. 1217–1229. Cited on page 138.

[232] E. A. LEE AND H. ZHENG, *Operational semantics of hybrid systems*, in Hybrid Systems: Computation and Control, vol. 3414, Springer Berlin Heidelberg, 2005, pp. 25–53. Series Title: Lecture Notes in Computer Science. Cited on pages 9 and 62.

[233] B. LEGAT, R. M. JUNGERS, AND P. A. PARRILO, *Generating Unstable Trajectories for Switched Systems via Dual Sum-Of-Squares Techniques*, in 19th International Conference on Hybrid Systems: Computation and Control, New York, New York, USA, 2016, ACM Press, pp. 51–60. Cited on pages 88, 98, 102, and 108.

[234] B. LEGAT, P. A. PARRILO, AND R. M. JUNGERS, *Certifying unstability of Switched Systems using Sum of Squares Programming*, tech. rep., Université catholique de Louvain, Oct. 2017. Cited on pages 88, 102, 107, 108, and 112.

[235] E. LELARASMEE, A. E. RUEHLI, AND A. L. SANGIOVANNI-VINCENTELLI, *The Waveform Relaxation Method for Time-Domain Analysis of Large Scale Integrated Circuits*, in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 1, 1982, pp. 131–145. Cited on pages 16, 51, and 145.

[236] D. LIBERZON, *Switching in Systems and Control*, Springer Science & Business Media, 2012. Cited on pages 116 and 125.

[237] 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. Cited on page 22.

[238] H. LIN AND P. J. ANTSAKLIS, *Stability and Stabilizability of Switched Linear Systems: A Survey of Recent Results*, IEEE Transactions on Automatic Control, 54 (2009), pp. 308–322. Cited on page 113.

[239] D. LIND AND B. MARCUS, *An Introduction to Symbolic Dynamics and Coding*, Cambridge university press, 1995. Cited on pages 103, 108, and 109.

[240] P. LINZ, *An Introduction to Formal Languages and Automata*, Jones & Bartlett Publishers, 2011. Cited on page 100.

[241] J. LU AND L. J. BROWN, *A multiple Lyapunov functions approach for stability of switched systems*, in Proceedings of the 2010 American Control Conference, IEEE, June 2010, pp. 3253–3256. Cited on page 116.

[242] J. LYGEROS, *Lecture notes on hybrid systems*, tech. rep., Department of Electrical and Computer Engineering University of Patras, 2004. Cited on page 60.

[243] N. LYNCH, R. SEGALA, AND F. VAANDRAGER, *Hybrid I/O automata*, Information and Computation, 185 (2003), pp. 105–157. Cited on page 64.

[244] M. MAESUMI, *An efficient lower bound for the generalized spectral radius of a set of matrices*, Linear Algebra and its Applications, 240 (1996), pp. 1–7. Cited on page 97.

[245] O. MALER AND G. BATT, *Approximating Continuous Systems by Timed Automata*, in Formal Methods in Systems Biology, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 77–89. Cited on page 22.

[246] O. MALER, Z. MANNA, AND A. PNUELI, *From timed to hybrid systems*, Real-Time: Theory in Practice, 600 (1992), pp. 447–484. Cited on pages 9 and 60.

[247] Z. MANNA AND A. PNUELI, *Verifying hybrid systems*, in Hybrid Systems SE - 2, vol. 736, Springer Berlin Heidelberg, 1993, pp. 4–35. Series Title: Lecture Notes in Computer Science. Cited on page 9.

[248] J. A. MARTINEZ, T. P. KURZWEG, S. P. LEVITAN, P. J. MARCHAND, AND D. M. CHIARULLI, *Mixed-Technology System-Level Simulation*, Analog Integrated Circuits and Signal Processing, 29 (2001), pp. 127–149. Cited on page 18.

[249] P. MAYRING, *Qualitative Content Analysis*, Forum: Qualitative Social Research, (2004), pp. 170–183. Cited on page 75.

[250] W. J. MCCALLA, *Fundamentals of Computer-Aided Circuit Simulation*, vol. 37, Springer Science & Business Media, 1987. Cited on page 17.

[251] B. MEYERS, J. DENIL, F. BOULANGER, C. HARDEBOLLE, C. JACQUET, AND H. VANGHELUWE, *A DSL for Explicit Semantic Adaptation*, in 7th International Workshop on Multi-Paradigm Modeling, Miami, United States, Sept. 2013, Springer, Berlin, Heidelberg, pp. 47–56. Series Title: CEUR Workshop Proceedings. Cited on pages 62, 137, 138, 143, 151, and 176.

[252] U. MIEKKALA AND O. NEVANLINNA, *Convergence of Dynamic Iteration Methods for Initial Value Problems*, SIAM Journal on Scientific and Statistical Computing, 8 (1987), pp. 459–482. Cited on page 16.

[253] D. MILLER AND J. THORPE, *SIMNET: The advent of simulator networking*, Proceedings of the IEEE, 83 (1995), pp. 1114–1123. Cited on pages 17 and 38.

[254] W. L. MIRANKER, *Parallel Methods for Approximating the Root of a Function*, IBM Journal of Research and Development, 13 (1969), pp. 297–301. Cited on page 16.

[255] ——, *A Survey of Parallelism in Numerical Analysis*, SIAM Review, 13 (1971), pp. 524–547. Cited on page 16.

[256] W. L. MIRANKER AND W. LINIGER, *Parallel methods for the numerical integration of ordinary differential equations*, Mathematics of Computation, 21 (1967), pp. 303–303. Cited on page 16.

[257] D. MITRA, *Asynchronous Relaxations for the Numerical Solution of Differential Equations by Parallel Processors*, SIAM Journal on Scientific and Statistical Computing, 8 (1987), pp. s43–s58. Cited on page 16.

[258] S. MITRA, D. LIBERZON, AND N. LYNCH, *Verifying average dwell time of hybrid systems*, ACM Transactions on Embedded Computing Systems, 8 (2008), pp. 1–37. Cited on page 116.

[259] M. MOORE AND J. WILHELMS, *Collision detection and response for computer animation*, in Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '88, New York, New York, USA, 1988, ACM Press, pp. 289–298. Cited on page 135.

[260] P. J. MOSTERMAN, *An Overview of Hybrid Simulation Phenomena and Their Support by Simulation Packages*, in Hybrid Systems: Computation and Control SE - 17, vol. 1569, Berg en Dal, The Netherlands, 1999, Springer Berlin Heidelberg, pp. 165–177. Series Title: Lecture Notes in Computer Science. Cited on pages 60, 62, 88, and 145.

[261] P. J. MOSTERMAN AND G. BISWAS, *A theory of discontinuities in physical system models*, Journal of the Franklin Institute, 335 (1998), pp. 401–439. Cited on page 63.

[262] P. J. MOSTERMAN AND G. BISWAS, *A comprehensive methodology for building hybrid models of physical systems*, Artificial Intelligence, 121 (2000), pp. 171–209. Cited on page 60.

[263] P. J. MOSTERMAN AND H. VANGHELUWE, *Computer Automated Multi-Paradigm Modeling: An Introduction*, Simulation, 80 (2004), pp. 433–450. Cited on page 2.

[264] W. MÜLLER AND E. WIDL, *Using FMI components in discrete event systems*, in Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), Seattle, WA, USA, 2015, pp. 1–6. Cited on page 62.

[265] J. R. MUNKRES, *Topology*, Prentice Hall, second ed., 2000. Cited on page 122.

[266] S. MUSTAFIZ, B. BARROCA, C. GOMES, AND H. VANGHELUWE, *Towards Modular Language Design Using Language Fragments: The Hybrid Systems Case Study*, in 13th International Conference on Information Technology - New Generations (ITNG), S. Latifi, ed., Las Vegas, NV USA, Apr. 2016, Springer, Cham, pp. 785–797. Cited on pages 8 and 61.

[267] S. MUSTAFIZ, C. GOMES, B. BARROCA, AND H. VANGHELUWE, *Modular Design of Hybrid Languages by Explicit Modeling of Semantic Adaptation*, in Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S

Symposium, Pasadena, California, Apr. 2016, IEEE, pp. 29:1–29:8. Series Title: DEVS '16. Cited on pages 8, 137, and 176.

[268] S. MUSTAFIZ AND H. VANGHELUWE, *Explicit Modelling of Statechart Simulation Environments*, in Summer Computer Simulation Conference, Vista, CA, 2013, Society for Modeling & Simulation International, pp. 21:1–21:8. Series Title: SCSC '13. Cited on page 12.

[269] A. MUZY, L. TOURAILLE, H. VANGHELUWE, O. MICHEL, M. K. TRAORÉ, AND D. R. C. HILL, *Activity Regions for the Specification of Discrete Event Systems*, in Spring Simulation Multiconference, San Diego, CA, USA, 2010, Society for Computer Simulation International, pp. 136:1–136:7. Cited on page 37.

[270] E. M. NAVARRO-LÓPEZ, *Hybrid-automaton models for simulating systems with sliding motion: Still a challenge*, IFAC Proceedings Volumes, 42 (2009), pp. 322–327. Cited on page 60.

[271] E. M. NAVARRO-LÓPEZ AND R. CARTER, *Hybrid automata: An insight into the discrete abstraction of discontinuous systems*, International Journal of Systems Science, 42 (2011), pp. 1883–1898. Cited on page 60.

[272] ——, *Deadness and how to disprove liveness in hybrid dynamical systems*, Theoretical Computer Science, 642 (2016), pp. 1–23. Cited on page 116.

[273] E. M. NAVARRO-LÓPEZ AND D. S. LAILA, *Group and Total Dissipativity and Stability of Multi-Equilibria Hybrid Automata*, IEEE Transactions on Automatic Control, 58 (2013), pp. 3196–3202. Cited on page 116.

[274] H. NEEMA, J. GOHL, Z. LATTMANN, J. SZTIPANOVITS, G. KARSAI, S. NEEMA, T. BAPTY, J. BATTEH, H. TUMMESCHEIT, AND C. SURESHKUMAR, *Model-based integration platform for FMI co-simulation and heterogeneous simulations of cyber-physical systems*, in 10th International Modelica Conference, 2014, pp. 10–12. Cited on page 61.

[275] A. R. NEWTON AND A. L. SANGIOVANNI-VINCENTELLI, *Relaxation-Based Electrical Simulation*, SIAM Journal on Scientific and Statistical Computing, 4 (1983), pp. 485–524. Cited on page 16.

[276] C. B. NIELSEN, P. G. LARSEN, J. FITZGERALD, J. WOODCOCK, AND J. PELESKA, *Systems of Systems Engineering: Basic Concepts, Model-Based Techniques, and Research Directions*, ACM Computing Surveys, 48 (2015), pp. 18:1–18:41. Place: New York, NY, USA. Cited on page 1.

[277] J. NIEVERGELT, *Parallel methods for integrating ordinary differential equations*, Communications of the ACM, 7 (1964), pp. 731–733. Cited on page 16.

[278] H. NILSSON, *Functional automatic differentiation with Dirac impulses*, ACM SIGPLAN Notices, 38 (2003), pp. 153–164. Place: New York, NY, USA. Cited on page 62.

[279] P. NILSSON AND N. OZAY, *On a Class of Maximal Invariance Inducing Control Strategies for Large Collections of Switched Systems*, in Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control, 2017, pp. 187–196. Cited on page 96.

[280] K. NORLING, D. BROMAN, P. FRITZSON, A. SIEMERS, AND D. FRITZSON, *Secure distributed co-simulation over wide area networks*, in 48th Conference on Simulation and Modelling, Göteborg, Sweden, 2007, pp. 14–23. Cited on page 37.

[281] M. NOWACK, J. ENDRIKAT, AND E. GUENTHER, *Review of Delphi-based scenario studies: Quality and design considerations*, Technological Forecasting and Social Change, 78 (2011), pp. 1603–1615. Cited on page 74.

[282] J. NUTARO, *Designing power system simulators for the smart grid: Combining controls, communications, and electro-mechanical dynamics*, in IEEE Power and Energy Society General Meeting, Detroit, MI, USA, July 2011, IEEE, pp. 1–5. Cited on page 61.

[283] ——, *A method for bounding error in multi-rate and federated simulations*, in Winter Simulation Conference, Washington, DC, USA, Dec. 2016, IEEE, pp. 967–976. Cited on page 62.

[284] J. NUTARO, P. T. KURUGANTI, V. PROTOPOPESCU, AND M. SHANKAR, *The split system approach to managing time in simulations of hybrid systems having continuous and discrete event components*, SIMULATION, 88 (2012), pp. 281–298. Cited on page 61.

[285] S. OH AND S. CHAE, *A Co-Simulation Framework for Power System Analysis*, Energies, 9 (2016), p. 131. Cited on page 58.

[286] C. OKOLI AND S. D. PAWLOWSKI, *The Delphi method as a research tool : An example , design considerations and applications*, Information & Management, 42 (2004), pp. 15–29. Cited on pages 74 and 75.

[287] M. OTTER AND H. ELMQVIST, *The DSblock model interface for exchanging model components*, in Proceedings of the Eurosim'95 Simulation Congress, 1995, pp. 505–510. Cited on pages 18 and 59.

[288] M. OTTER, M. MALMHEDEN, H. ELMQVIST, S. E. MATTSSON, AND C. JOHNSSON, *A New Formalism for Modeling of Reactive and Hybrid Systems*, in 7th International Modelica Conference, Como, Italy, Oct. 2009, Linköping University Electronic Press; Linköpings universitet, pp. 364–377. Cited on pages 138 and 176.

[289] P. PALENSKY, A. A. VAN DER MEER, C. D. LOPEZ, A. JOSEPH, AND K. PAN, *Cosimulation of Intelligent Power Systems: Fundamentals, Software Architecture, Numerics, and Coupling*, IEEE Industrial Electronics Magazine, 11 (2017), pp. 34–50. Cited on page 27.

[290] A. PAPACHRISTODOULOU AND S. PRAJNA, *A tutorial on sum of squares techniques for systems analysis*, in American Control Conference, Portland, OR, USA, 2005, IEEE, pp. 2686–2700. Cited on page 136.

[291] K. PARK, C. FELIPPA, AND J. DERUNTZ, *Stabilization of staggered solution procedures for fluid-structure interaction analysis*, Computational methods for fluid-structure interaction problems, 26 (1977), p. 51. Cited on page 16.

[292] P. A. PARRILO AND A. JADBABAIE, *Approximation of the Joint Spectral Radius of a Set of Matrices Using Sum of Squares*, in Hybrid Systems: Computation and

Control, Pisa, Italy, 2007, Springer, Berlin, Heidelberg, pp. 444–458. Cited on pages 103 and 113.

[293] ——, *Approximation of the joint spectral radius using sum of squares*, Linear Algebra and its Applications, 428 (2008), pp. 2385–2402. Cited on page 97.

[294] H. M. PAYNTER, *Analysis and Design of Engineering Systems*, MIT press, 1961. Cited on pages 54 and 71.

[295] N. PEDERSEN, T. BOJSEN, AND J. MADSEN, *Co-simulation of Cyber Physical Systems with HMI for Human in the Loop Investigations*, in Symposium on Theory of Modeling & Simulation, Virginia Beach, Virginia, USA, 2017, Society for Computer Simulation International, pp. 1:1–1:12. Series Title: TMS/DEVS '17. Cited on page 2.

[296] N. PEDERSEN, T. BOJSEN, J. MADSEN, AND M. VEJLGAARD-LAURSEN, *FMI for Co-Simulation of Embedded Control Software*, in The First Japanese Modelica Conferences, Tokyo, Japan, May 2016, Linköping University Electronic Press, pp. 70–77. Cited on page 62.

[297] N. PEDERSEN, K. LAUSDAHL, E. VIDAL SANCHEZ, P. G. LARSEN, AND J. MADSEN, *Distributed Co-Simulation of Embedded Control Software with Exhaust Gas Recirculation Water Handling System using INTO-CPS*, in 7th International Conference on Simulation and Modeling Methodologies, Technologies and Applications, SCITEPRESS - Science and Technology Publications, 2017, pp. 73–82. Cited on pages 6, 8, 22, 24, and 66.

[298] N. C. PETRELLIS, A. N. BIRBAS, M. K. BIRBAS, E. P. MARIATOS, AND G. D. PAPADOPOULOS, *Simulating Hardware, Software and Electromechanical Parts Using Communicating Simulators*, Design Automation for Embedded Systems, 3 (1998), pp. 187–198. Cited on page 18.

[299] S. PETTERSSON, *Synthesis of switched linear systems*, in 42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475), vol. 5, IEEE, 2003, pp. 5283–5288. Cited on page 113.

[300] M. PHILIPPE, R. ESSICK, G. E. DULLERUD, AND R. M. JUNGERS, *Stability of discrete-time switching systems with constrained switching sequences*, Automatica, 72 (2016), pp. 242–250. Cited on pages 105 and 121.

[301] R. PLATEAUX, J. CHOLEY, O. PENAS, AND A. RIVIERE, *Towards an integrated mechatronic design process*, in International Conference on Mechatronics, vol. 00, Malaga, Spain, 2009, IEEE, pp. 1–6. Cited on page 1.

[302] A. PLATZER, *Logical Analysis of Hybrid Systems*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. Publication Title: Descriptional Complexity of Formal Systems. Cited on page 64.

[303] U. POHLMANN, W. SCHÄFER, H. REDDEHASE, J. RÖCKEMANN, AND R. WAGNER, *Generating Functional Mockup Units from Software Specifications*, in 9th International MODELICA Conference, Munich, Germany, Nov. 2012, Linköping University Electronic Press; Linköpings universitet, pp. 765–774. Cited on page 176.

[304] E. POSSE, J. DE LARA, AND H. VANGHELUWE, *Processing causal block diagrams with graphgrammars in atom3*, in Workshop on Applied Graph Transformation (AGT), Grenoble, France, Apr. 2002, Springer, Berlin, Heidelberg, pp. 23–34. Cited on page 141.

[305] C. POWELL, *The Delphi Technique: Myths and realities*, Methodological Issues in Nursing Research, 41 (2003), pp. 376–382. Cited on page 74.

[306] P. PRABHAKAR AND M. GARCÍA SOTO, *Formal Synthesis of Stabilizing Controllers for Switched Systems*, in Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control, 2017, pp. 111–120. Cited on page 113.

[307] S. M. PRABHU AND P. J. MOSTERMAN, *Modeling, Simulating, and Validating a Power Window System Using a Model-Based Design Approach*. Cited on page 147.

[308] C. PTOLEMAEUS, *System Design, Modeling, and Simulation: Using Ptolemy II*, Berkeley: Ptolemy.org, 2014. Cited on pages 29 and 61.

[309] D. QUAGLIA, R. MURADORE, R. BRAGANTINI, AND P. FIORINI, *A SystemC/Matlab co-simulation tool for networked control systems*, Simulation Modelling Practice and Theory, 23 (2012), pp. 71–86. Cited on page 61.

[310] G. QUESNEL, R. DUBOZ, D. VERSMISSE, AND E. RAMAT, *DEVS coupling of spatial and ordinary differential equations: VLE framework*, in Open International Conference on Modeling and Simulation, vol. 5, Citeseer, June 2005, pp. 281–294. Cited on pages 61 and 146.

[311] C. RACKAUCKAS AND Q. NIE, *Differentialequations. jl–a performant and feature-rich ecosystem for solving differential equations in Julia*, Journal of Open Research Software, 5 (2017). Cited on page 133.

[312] M. RADETZKI AND R. S. KHALIGH, *Accuracy-adaptive Simulation of Transaction Level Models*, in Proceedings of the Conference on Design, Automation and Test in Europe, New York, NY, USA, 2008, ACM, pp. 788–791. Series Title: DATE '08. Cited on pages 19 and 88.

[313] C. ROBINSON, *Dynamical Systems: Stability, Symbolic Dynamics, and Chaos*, CRC press, 1998. Cited on page 130.

[314] G. C. ROTA AND W. STRANG, *A note on the joint spectral radius*, in Proceedings of the Netherlands Academy 22, 1960, pp. 379–381. Cited on page 96.

[315] J. A. ROWSON, *Hardware/Software Co-Simulation*, in 31st Conference on Design Automation, 1994, pp. 439–440. Cited on page 18.

[316] L. SACHS, *Angewandte Statistik*, Springer-Verlag, Berlin Heidelberg, 1997. Cited on page 75.

[317] S. SADJINA, L. T. KYLLINGSTAD, S. SKJONG, AND E. PEDERSEN, *Energy conservation and power bonds in co-simulations: Non-iterative adaptive step size control and error estimation*, Engineering with Computers, 33 (2017), pp. 607–620. Cited on pages 54, 83, 183, and 194.

[318] S. SADJINA AND E. PEDERSEN, *Energy Conservation and Coupling Error Reduction in Non-Iterative Co-Simulations*, tech. rep., SINTEF Alesund, June 2016. Cited on pages 58, 71, and 174.

[319] S. E. SAIDI, N. PERNET, Y. SOREL, AND A. B. KHALED, *Acceleration of FMU Co-Simulation On Multi-core Architectures*, in The First Japanese Modelica Conferences, Tokyo, Japan, May 2016, Linköping University Electronic Press, pp. 106–112. Cited on page 83.

[320] R. A. SALEH, S.-J. JOU, AND A. R. NEWTON, *Mixed-Mode Simulation and Analog Multilevel Simulation*, vol. 279, Springer Science & Business Media, 2013. Cited on page 18.

[321] D. SCHRAMM, W. LALO, AND M. UNTERREINER, *Application of Simulators and Simulation Tools for the Functional Design of Mechatronic Systems*, Solid State Phenomena, 166-167 (2010), pp. 1–14. Cited on page 1.

[322] G. SCHWEIGER, G. ENGEL, J. SCHOEGGL, I. HAFNER, C. GOMES, AND T. NOUIDUI, *Co-Simulation - an Empirical Survey: Applications, Recent Developments and Future Challenges*, in MATHMOD 2018 Extended Abstract Volume, Vienna, Austria, 2018, ARGESIM Publisher Vienna, pp. 125–126. Cited on page 74.

[323] G. SCHWEIGER, C. GOMES, G. ENGEL, I. HAFNER, J. SCHOEGGL, A. POSCH, AND T. NOUIDUI, *Functional Mock-up Interface: An empirical survey identifies research challenges and current barriers*, in The American Modelica Conference, Cambridge, MA, USA, 2018, Linköping University Electronic Press, Linköpings universitet, pp. 138–146. Cited on pages xi, 74, 78, and 81.

[324] SCHWEIZER AND D. LU, *Predictor/corrector co-simulation approaches for solver coupling with algebraic constraints*, ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik, 95 (2015), pp. 911–938. Cited on pages 46, 47, 51, 54, and 174.

[325] B. SCHWEIZER, P. LI, AND D. LU, *Explicit and Implicit Cosimulation Methods: Stability and Convergence Analysis for Different Solver Coupling Approaches*, Journal of Computational and Nonlinear Dynamics, 10 (2015), p. 051007. Cited on pages 54, 57, 91, 113, and 179.

[326] B. SCHWEIZER, P. LI, D. LU, AND T. MEYER, *Stabilized implicit co-simulation methods: Solver coupling based on constitutive laws*, Archive of Applied Mechanics, 85 (2015), pp. 1559–1594. Cited on pages 113 and 174.

[327] B. SCHWEIZER AND D. LU, *Semi-implicit co-simulation approach for solver coupling*, Archive of Applied Mechanics, 84 (2014), pp. 1739–1769. Cited on page 174.

[328] ——, *Stabilized index-2 co-simulation approach for solver coupling with algebraic constraints*, Multibody System Dynamics, 34 (2015), pp. 129–161. Cited on page 174.

[329] B. SCHWEIZER, D. LU, AND P. LI, *Co-simulation method for solver coupling with algebraic constraints incorporating relaxation techniques*, Multibody System Dynamics, 36 (2016), pp. 1–36. Cited on pages 43, 47, 51, 54, 144, and 174.

[330] L. F. SHAMPINE AND M. W. REICHELT, *The matlab ode suite*, SIAM journal on scientific computing, 18 (1997), pp. 1–22. Cited on page 89.

[331] R. SHORTEN, F. WIRTH, O. MASON, K. WULFF, AND C. KING, *Stability Criteria for Switched and Hybrid Systems*, SIAM Review, 49 (2007), pp. 545–592. Cited on page 110.

[332] S. SICKLINGER, V. BELSKY, B. ENGELMANN, H. ELMQVIST, H. OLSSON, R. WÜCHNER, AND K. U. BLETZINGER, *Interface Jacobian-based Co-Simulation*, International Journal for Numerical Methods in Engineering, 98 (2014), pp. 418–444. Cited on pages 47, 145, and 174.

[333] M. SIPSER, *Introduction to the Theory of Computation*, Thomson Course Technology, 2013. Cited on page 104.

[334] C. SLOTH AND R. WISNIEWSKI, *Robust stability of switched systems*, in 53rd IEEE Conference on Decision and Control, Los Angeles, CA, USA, Dec. 2014, IEEE, pp. 4685–4690. Cited on page 113.

[335] J. SOMERVILLE, *Critical Factors Affecting the Assessment of Student Learning Outcomes: A Delphi Study of the Opinions of Community College Personnel*, Journal of Applied Research in the Community College, (2008). Cited on page 74.

[336] M. SPIEGEL, P. REYNOLDS, AND D. BROGAN, *A Case Study of Model Context for Simulation Composability and Reusability*, in Proceedings of the Winter Simulation Conference, 2005., vol. 2005, IEEE, 2005, pp. 437–444. Cited on pages 7, 79, and 83.

[337] H. STACHOWIAK, *Allgemeine Modelltheorie*, Springer-Verlag, Wien and New York, 1973. Cited on page 6.

[338] G. STETTINGER, M. BENEDIKT, M. HORN, J. ZEHETNER, AND C. GIEBENHAIN, *Control of a magnetic levitation system with communication imperfections: A model-based coupling approach*, Control Engineering Practice, 58 (2017), pp. 161–170. Cited on page 43.

[339] G. STETTINGER, M. HORN, M. BENEDIKT, AND J. ZEHETNER, *Model-based coupling approach for non-iterative real-time co-simulation*, in European Control Conference (ECC), Strasbourg, France, 2014, IEEE, pp. 2084–2089. Cited on pages 43, 59, and 174.

[340] G. STETTINGER, J. ZEHETNER, M. BENEDIKT, AND N. THEK, *Extending Co-Simulation to the Real-Time Domain*, in SAE Technical Paper, Apr. 2013. Cited on page 59.

[341] A. STUART AND A. R. HUMPHRIES, *Dynamical Systems and Numerical Analysis*, vol. 2, Cambridge University Press, 1998. Cited on pages 94 and 119.

[342] Z. SUN AND S. GE, *Analysis and synthesis of switched linear control systems*, Automatica, 41 (2005), pp. 181–195. Cited on page 113.

[343] J. M. SUSSMAN, *Collected views on complexity in systems*, tech. rep., Massachusetts Institute of Technology. Engineering Systems Division, 2002. Cited on page 6.

[344] R. TARJAN, *Depth-first search and linear graph algorithms*, in 12th Annual Symposium on Switching and Automata Theory (Swat 1971), vol. 1, East Lansing, MI, USA, Oct. 1971. Cited on page 51.

[345] 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, Sept. 2016, IEEE, pp. 1–5. Cited on pages 61, 62, and 64.

[346] B. TAYLOR, *Methodus Incrementorum Directa et Inversa*, London, (1715). Cited on page 39.

[347] M. TESCHNER, S. KIMMERLE, B. HEIDELBERGER, G. ZACHMANN, L. RAGHUPATHI, A. FUHRMANN, M.-P. CANI, F. FAURE, N. MAGNENAT-THALMANN, W. STRASSER, AND P. VOLINO, *Collision Detection for Deformable Objects*, Computer Graphics Forum, 24 (2005), pp. 61–81. Cited on page 135.

[348] C. THULE, C. GOMES, J. DEANTONI, P. G. LARSEN, J. BRAUER, AND H. VANGHELUWE, *Towards Verification of Hybrid Co-simulation Algorithms*, in 2nd Workshop on Formal Co-Simulation of Cyber-Physical Systems, Toulouse, France, 2018, Springer, Cham, p. to be published. Cited on pages 67 and 83.

[349] C. THULE AND P. LARSEN, *Investigating Concurrency in the Co-Simulation Orchestration Engine for INTO-CPS*, Proceedings of the Institute for System Programming of the RAS, 28 (2016), pp. 139–156. Cited on page 83.

[350] C. THULE, K. LAUSDAHL, C. GOMES, G. MEISL, AND P. G. LARSEN, *Maestro: The INTO-CPS Co-simulation Framework*, Simulation Modelling Practice and Theory, 92 (2019), pp. 45–61. Cited on page 23.

[351] T. TOMIYAMA, V. D'AMELIO, J. URBANIC, AND W. ELMARAGHY, *Complexity of Multi-Disciplinary Design*, CIRP Annals - Manufacturing Technology, 56 (2007), pp. 185–188. Cited on pages 1 and 8.

[352] M. TRCKA, M. WETTER, AND J. HENSEN, *Comparison of co-simulation approaches for building and HVAC/R system simulation*, in International IBPSA Conference, Beijing, China, 2007. Cited on page 51.

[353] S. TRIPAKIS, *Bridging the semantic gap between heterogeneous modeling formalisms and FMI*, in International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), Samos, Greece, July 2015, IEEE, pp. 60–69. Cited on pages 61, 62, 138, and 176.

[354] F. TSENG AND G. HULBERT, *Network-distributed multibody dynamics simulation—gluing algorithm*, Advances in Computational Multibody Dynamics, (1999), pp. 521–540. Cited on page 18.

[355] A. TUKKER, *Product services for a resource-efficient and circular economy – a review*, Journal of Cleaner Production, 97 (2015), pp. 76–91. Cited on page 20.

[356] S. UCHITEL AND D. YANKELEVICH, *Enhancing architectural mismatch detection with assumptions*, in Engineering of Computer Based Systems, 2000. (ECBS 2000)

Proceedings. Seventh IEEE International Conference and Workshopon The, 2000, pp. 138–146. Cited on page 8.

[357] A. M. UHRMACHER, *Variable structure models: Autonomy and control answers from two different modeling approaches*, in AI, Simulation and Planning in High Autonomy Systems, IEEE Comput. Soc. Press, 1993, pp. 133–139. Cited on page 37.

[358] B. VAN ACKER, J. DENIL, P. D. MEULENAERE, AND H. VANGHELUWE, *Generation of an Optimised Master Algorithm for FMI Co-simulation*, in Symposium on Theory of Modeling & Simulation-DEVS Integrative, Alexandria, Virginia, USA, Apr. 2015, Society for Computer Simulation International San Diego, CA, USA, pp. 946–953. Cited on pages 45, 66, 138, 144, 145, 177, and 182.

[359] H. VAN DER AUWERAER, J. ANTHONIS, S. DE BRUYNE, AND J. LEURIDAN, *Virtual engineering at work: The challenges for designing mechatronic products*, Engineering with Computers, 29 (2013), pp. 389–408. Cited on pages 1, 6, 8, 19, and 20.

[360] A. J. VAN DER SCHAFT AND J. M. SCHUMACHER, *An Introduction to Hybrid Dynamical Systems*, vol. 251, Springer London, 2000. Cited on page 63.

[361] S. VAN MIERLO, *Explicitly Modelling Model Debugging Environments*, in ACM Student Research Competition (ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems), CEUR, 2015, pp. 24–29. Cited on page 12.

[362] Y. VAN TENDELOO AND H. VANGHELUWE, *Activity in PythonPDEVS*, ITM Web of Conferences, 3 (2014), p. 10. Cited on page 37.

[363] ——, *PythonPDEVS: A distributed Parallel DEVS simulator*, in Spring Simulation Multiconference, Alexandria, Virginia, 2015, Society for Computer Simulation International, pp. 844–851. Series Title: SpringSim '15. Cited on page 37.

[364] ——, *Increasing the performance of a Discrete Event System Specification simulator by means of computational resource usage "activity" models*, SIMULATION, 93 (2017), pp. 1045–1061. Cited on page 190.

[365] ——, *An Introduction to Classic DEVS*, tech. rep., University of Antwerp, 2017. Cited on pages 29 and 33.

[366] H. VANGHELUWE, *DEVS as a common denominator for multi-formalism hybrid systems modelling*, in International Symposium on Computer-Aided Control System Design (Cat. No.00TH8537), Anchorage, AK, USA, 2000, IEEE, pp. 129–134. Cited on pages 29 and 61.

[367] ——, *Foundations of Modelling and Simulation of Complex Systems*, Electronic Communications of the EASST, 10 (2008). Cited on pages 6 and 9.

[368] H. VANGHELUWE, J. DE LARA, AND P. J. MOSTERMAN, *An introduction to multi-paradigm modelling and simulation*, in AI, Simulation and Planning in High Autonomy Systems, SCS, 2002, pp. 9–20. Cited on pages 6 and 189.

[369] H. L. VANGHELUWE, G. C. VANSTEENKISTE, AND E. J. KERCKHOFFS, *Simulation for the Future : Progress of the Esprit Basic Research Working Group 8467*, in

Proceedings of the 1996 European Simulation Symposium, Genoa, 1996, Society for Computer Simulation International, pp. XXIX–XXXIV. Cited on page 18.

[370] K. VANHERPEN, J. DENIL, P. DE MEULENAERE, AND H. VANGHELUWE, *Design-Space Exploration in Model Driven Engineering: An Initial Pattern Catalogue*, in 1st International Workshop on Combining Modelling with Search and Example-Based Approaches (CMSEBA), CEUR Workshop Proceedings (Vol-1340), Sept. 2014, pp. 42–51. Cited on page 190.

[371] M. E. VARA LARSEN, J. DE ANTONI, B. COMBEMALE, AND F. MALLET, *A Behavioral Coordination Operator Language (BCOoL)*, in 18th International Conference on Model Driven Engineering Languages and Systems (MODELS), Ottawa, ON, Canada, Sept. 2015, IEEE, pp. 186–195. Cited on page 176.

[372] A. VERHOEVEN, B. TASIC, T. G. J. BEELEN, E. J. W. TER MATEN, AND R. M. M. MATTHEIJ, *BDF compound-fast multirate transient analysis with adaptive stepsize control*, Journal of numerical analysis, Industrial and Applied Mathematics, 3 (2008), pp. 275–297. Cited on page 54.

[373] G. WANNER AND E. HAIRER, *Solving Ordinary Differential Equations I: Nonstiff Problems*, vol. 1, Springer-Verlag, springer s ed., 1991. Cited on pages 53, 58, and 97.

[374] D. B. WEST, *Introduction to Graph Theory*, vol. 2, Prentice hall Upper Saddle River, 2001. Cited on page 108.

[375] M. WETTER, *Co-simulation of building energy and control systems with the Building Controls Virtual Test Bed*, Journal of Building Performance Simulation, 4 (2010), pp. 185–203. Cited on pages 45 and 174.

[376] E. WIDL, W. MÜLLER, A. ELSHEIKH, M. HÖRTENHUBER, AND P. PALENSKY, *The FMI++ library: A high-level utility package for FMI for model exchange*, in Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), Berkeley, CA, USA, 2013, IEEE, pp. 1–6. Cited on page 61.

[377] N. WIRTH, *Extended backus-naur form (EBNF)*, 1996. Pages: 2996 Publication Title: ISO/IEC. Cited on page 166.

[378] M.-C. WU AND M.-C. SHIH, *Simulated and experimental study of hydraulic anti-lock braking system using sliding-mode PWM control*, Mechatronics, 13 (2003), pp. 331–351. Cited on page 2.

[379] X. XU AND P. ANTSAKLIS, *Optimal Control of Switched Systems Based on Parameterization of the Switching Instants*, Automatic Control, 49 (2004), pp. 2–16. Cited on page 113.

[380] X. XU AND P. J. ANTSAKLIS, *Optimal control of switched systems via non-linear optimization based on direct differentiations of value functions*, International Journal of Control, 75 (2002), pp. 1406–1426. Cited on page 113.

[381] T. YANG, *Networked control system: A brief survey*, IEE Proceedings - Control Theory and Applications, 153 (2006), pp. 403–412. Cited on pages 117 and 135.

[382] F. YILMAZ, U. DURAK, K. TAYLAN, AND H. OĞUZTÜZÜN, *Adapting Functional Mockup Units for HLA-compliant Distributed Simulation*, in 10th International Modelica Conference, 2014. Cited on page 61.

[383] B. P. ZEIGLER, *Theory of Modelling and Simulation*, New York, Wiley, 1976. Cited on pages 6, 29, and 38.

[384] ——, *Embedding DEV&DESS in DEVS*, in DEVS Integrative Modeling & Simulation Symposium, vol. 7, 2006. Cited on pages 61 and 64.

[385] B. P. ZEIGLER AND J. S. LEE, *Theory of quantized systems: Formal basis for DEVS/HLA distributed simulation environment*, in Enabling Technology for Simulation Science II, vol. 3369, Orlando, FL, United States, Aug. 1998, SPIE 3369, pp. 49–58. Cited on page 62.

[386] B. P. ZEIGLER, H. PRAEHOFER, AND T. G. KIM, *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*, Academic press, 2 ed., 2000. Cited on pages 32, 34, 63, and 64.

[387] G. ZHAI, B. HU, K. YASUDA, AND A. N. MICHEL, *Piecewise Lyapunov Functions for switched systems with average dwell time*, Asian Journal of Control, 2 (2000), pp. 192–197. Cited on page 116.

[388] F. ZHANG, M. YEDDANAPUDI, AND P. J. MOSTERMAN, *Zero-Crossing Location and Detection Algorithms For Hybrid System Simulation*, in IFAC Proceedings Volumes, vol. 41, Seoul, Korea, July 2008, Elsevier Ltd, pp. 7967–7972. Cited on pages 62, 115, 117, and 145.

[389] W. ZHANG, A. ABATE, J. HU, AND M. P. VITUS, *Exponential stabilization of discrete-time switched linear systems*, Automatica, 45 (2009), pp. 2526–2536. Cited on page 113.

[390] X. ZHAO, L. ZHANG, P. SHI, AND M. LIU, *Stability and Stabilization of Switched Linear Systems With Mode-Dependent Average Dwell Time*, Automatic Control, 57 (2012), pp. 1809–1815. Cited on page 113.

[391] V. ŽIVOJNOVIC AND H. MEYR, *Compiled HW/SW Co-simulation*, in Proceedings of the 33rd Annual Design Automation Conference, New York, NY, USA, 1996, ACM, pp. 690–695. Series Title: DAC '96. Cited on page 18.

[392] M. ZWOLINSKI, C. GARAGATE, Z. MRCARICA, T. J. KAZMIERSKI, AND A. D. BROWN, *Anatomy of a simulation backplane*, IEE Proceedings - Computers and Digital Techniques, 142 (1995), pp. 377–385(8). Cited on page 18.