<http://www.diva-portal.org>

Preprint

This is the submitted version of a paper presented at *1st Workshop on Formal Co-Simulation of Cyber-Physical Systems A satellite event of SEFM2017- 15th International conference on Software Engineering and Formal Methods*.

Citation for the original published paper:

Jinzhi, L., Törngren, M., Chen, D-J., Wang, J. (2017)

A Tool Integration Language to Formalize Co-simulation Tool-chains for Cyber-physical System (CPS).

In:

N.B. When citing this work, cite the original published paper.

Permanent link to this version:

<http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-211291>

A Tool Integration Language to Formalize Co-simulation Tool-chains for Cyber-physical System (CPS)

Jinzhi Lu^{1*}, Martin Törngren¹, De-Jiu Chen¹, and Jian Wang²

¹ KTH Royal Institute of Technology,
Brinellvgen 83, SE- 100 44 Stockholm, Sweden
{jinzh1,martint}@kth.se

chen@md.kth.se

<https://www.kth.se/en/itm/inst/mmk/avdelningar/mda>

² University of Electronic Science and Technology of China,
Xiyuan Ave, West Hi-Tech Zone, 611731, Chengdu, China.
wangjian3630@uestc.edu.cn

Abstract. Co-simulation has grown from point-to-point between simulation tools for specific purposes to complex tool-chains which often require additional functionalities, e.g., process management, data management and tool integration. With these additional functionalities, the related design activities could be controlled and implemented by unified platforms to improve efficiency and effectiveness. Due to increasing complexity and size of co-simulation tool-chains, a systematic approach is needed to formalize their evolution in order to analyze functionalities and evaluate their structures before development. In this paper, we extend a proposed domain specific language, - named Tool Integration Language (TIL) - to describe co-simulation tool-chain architectures on a high abstraction level aiming to promote the efficiency and effectiveness of co-simulation tool-chain development by the use of Model-based System Engineering (MBSE). We introduce how the extended TIL formalizes structures and present two industrial cases of co-simulation tool-chain from previous experiences and describe them using the TIL. Finally, we conclude this paper and introduce future work -a further extension of TIL supporting MBSE tool-chain development.

Keywords: Domain Specific Language, Tool Integration, Co-simulation

1 Introduction

Co-simulation is a term used for creating combined simulations across multiple domains. A typical example where co-simulation is strongly motivated is in Cyber-Physical Systems (CPS), involving domains such as embedded systems,

* Jinzhi Lu is a Ph.D. student in Mechatronics, School of Industrial Engineering and Management.

controls and computing and so on. It makes use of domain models and hardware, e.g. executing programs [4], hydraulic models [23] and multibody dynamics models [9] to support verification and validation of integrated system from system views. These models and hardware represent dynamic behaviors of specific subsystems and are integrated by a co-simulation environment.

Co-simulation is thus a natural technique in order to deal with the increasing complexities of CPS [12] to support concurrent, distributed and collaborative design during CPS development. It has the potential to resolve the problems of isolation among hardware and models from different tools and allows subsystem developers to build specific models using their own tools. System developers implement co-simulation to make use of such models to predict global behaviors in the initial phase of concept design.

Co-simulation for CPS has made progress through the introduction of the FMI, a tool independent standard that supports both model exchange and co-simulation [25]. However, the increasing complexity poses additional challenges both within and beyond co-simulation:

- **Technical challenges to co-simulation** - concentrates on the technical issues of co-simulation. In [13], the author provides several challenges which could influence *accuracy* and *robustness* of co-simulation implementations.
- **Information challenges** - focuses on the system aspects of CPS and integration of data, model, tool and information. Increasing CPS complexities with more complex requirements, system structures and verification & validation activities results in more requirements, such as *traceability* [2] and *consistency* [33]. The increasing number of stakeholders participating in the CPS design process leads to more design elements required, -e.g. tools, models and data-, which means tool-chains need more functionalities to manage and integrate such design elements.
- **Process challenges** - refers to the challenges of process management of co-simulation.
- **Social challenges** - refers to integration of stakeholders' social network. For example, during large-scale system development, different platforms supporting co-simulation for subsystems also need to be integrated in a unified framework or architecture in order to develop more complex large-scale system.

These challenges lead tool-chain developers to extend the co-simulation tool-chains' functionalities, which results in increasing complexity of tool-chains. This in turn challenges developers to analyze functionalities of co-simulation tool-chains and to make decisions about the techniques used to realize functionalities in tool-chain development. Traditional approaches for co-simulation tool-chains cannot satisfy the demand of stakeholders to develop co-simulation tool-chains in an efficient and effective way. The reasons include:

- Document based requirements are insufficient for formalizing the functionalities needed in co-simulation tool-chains due to the increasing complexities.

- In each simulation tool, adapters used for co-simulation, - referring to a software supporting simulation tools generating their co-simulation interfaces with other tools, are often developed by their developers themselves which increases R&D cost of the co-simulation tool-chains.
- Traditional tool-chain development used for developers is to obtain requirements from their customers by meetings and documents and validate their tool-chains using industrial practices of their customers. However, this approach takes a long time and changes during validation could cause risks for exceeding R&D cost and project delay.

In order to deal with the increasing complexity of co-simulation tool-chains, we proposed a methodology [16] that aims to make use of a model-based approach to formalize their architectures based on a system engineering approach, -called Model-based Systems Engineering (MBSE). It could support developers in analyzing and making decisions about co-simulation tool-chains' functionalities during development. In our methodology, we adopt and extend a domain specific language, -called Tool Integration Language (TIL) [5]-, to represent the architectures of a co-simulation tool-chain at a high level of abstraction. The original TIL aims to formalize tool-integration of tool-chains. We adopt and extend it, formalizing architectures of co-simulation tool-chain using IEEE 1471[15]³ and supporting improved description of the co-simulation features of tool-chains. Based on the extended TIL, developers can formalize the co-simulation tool-chains to analyze their functionalities and evaluate their structures before development.

This paper is structured as follows. *Section 2* introduces a basic concept of tool integration during co-simulation. Based on the analysis, we highlight possibilities to formalize the co-simulation tool-chains by the use of an extended TIL which is presented in *Section 3*. In *Section 4*, we illustrate how to use TIL on two case studies from previous research. We conclude and suggest future work in *Section 5*.

2 Tool Integration for Co-simulation

2.1 A Systems Engineering Approach to Developing Co-simulation Tool-chains

In previous research [16], we proposed a systems engineering approach for MBSE tool-chain development. We make use of IEEE 1471 to describe the architectures of MBSE tool-chains. Stakeholders of co-simulation tool-chains are surveyed to obtain viewpoints from the social layer, process layer, information layer and technical layer. These viewpoints then are used to construct the views to address the functionalities of tool-chain. These views are realized by techniques, -combined

³ IEEE 1471 is the predecessor of ISO/IEC 42010 [30]. We use IEEE 1471 in this paper, because IEEE 1471 is light weight and its overview of architecture description is sufficient to describe co-simulation tool-chain development.

with environment policy constraints to select tools and tool interactions-, in co-simulation tool-chains. In closing, tools and tool interactions construct co-simulation tool-chains.

In order to make use of an MBSE approach to analyze dependencies [28] and tool-integration of co-simulation tool-chains, two kinds of references models are provided in Table 1. The reference models are based on IEEE 1471 and support tool-chain development which propose a generic method to describe the dependencies of tool-chain development (From stakeholders to MBSE tool-chains' structure). The tool integration reference models are proposed to describe logic flows of co-simulation tool-chains.

Table 1. References Model for dependency and tool integration

Dependency	
Reference model	Description
<i>Stakeholders</i>	refer to people related to co-simulation tool-chains, e.g., users, developers.
<i>Viewpoints</i>	refer to a specifications of rulers to cover stakeholders' concerns about co-simulation tool-chains.
<i>Views</i>	refer to functional requirements of co-simulation tool-chains from the perspective of a related set of viewpoints.
<i>Technique</i>	refer to technical solutions used for realizing views.
<i>Environmentrules</i>	refer to environment rules and constraints of tool-chain development.
<i>Tool&Tool interactions</i>	refer tools and tool interactions used in co-simulation tool-chains.
Tool integration	
Reference model	Description
<i>Technique Space</i>	refer to a unified representation of data, tool operations, co-simulation models
<i>Tool adapter</i>	refer to integrated tools or interfaces exposing both functionalities and data. It includes <i>tool interface</i> [5], <i>integration interface</i> [5] and logic that translates between the interfaces.

In this paper, we extend TIL to formalize the co-simulation tool-chains' structure and identify further work to extend TIL to the reference models in Table 1. After first identifying the basic elements occurring during tool integration of co-simulation, we propose several approaches to identify high-level abstractions in the structures. We then use the high-level abstractions, which are common to all integration styles to construct the tool integration language.

2.2 Definitions and Scope

This section defines important terms used throughout the paper. Based on the definition of the MBSE tool-chain proposed in our previous paper, [16], we specified a definition of the **co-simulation tool-chain**: more than one modeling tool and hardware that, when combined, can support and implement a co-simulation workflow. Tool integration of co-simulation tool-chains has several dimensions; in this paper, we concentrate on:

- ☐ **Control integration** - focuses on the ability of tools to perform notifications and activate the tools under program control, [38].
- ☐ **Data integration** - refers to ensuring that all the development information in the design environment is managed as a consistent whole, [31].

- **Co-simulation** - refers to run-time communication between software and hardware or between software during co-simulation ([29],[39]). We extend this definition in TIL in order to describe the co-simulation features.

In related work on co-simulation (see further section 2.3), tool integration is generally described in the following three ways: (a) **Control-centric approaches** focus on of service invocations, application programming interfaces (API) and tool operations. (b) **Data-centric approaches** which can be specified in two ways: (b.1) Development information exchange and reuse by transferring data between tools. (b.2) Linking and tracing of development information without operations of data. (c) **Co-simulation approaches** focus on run-time communications between simulation tools or between simulation tools and hardware for dynamic behavior analysis.

In our proposed methodology in [16], we define a MBSE tool-chain development process as a mission in IEEE 1471. The system in the architecture is the target MBSE tool-chains. In IEEE 1471, at least four kinds of stakeholders and their viewpoints need to be considered: developers, users, maintainers and acquirers.

2.3 Tool integration method

A considerable number of methods for tool integration have been used in co-simulation environments. In order to analyze these methods and identify high-level abstractions, we analyze literature reviews in [13] and classify the methods with examples:

- **Integrated Data model:** Authors designed data models, -ontology, in specific tools which could be mapped to elements in the co-simulation process to control execution [2, 36, 37].
- **Integrated Process:** Tools are used to describe the workflow of co-simulations [32, 41].
- **Service-oriented approach:** In [35], a web based HLA federate was developed.
- **Tool-centric:** Specific tools are used to control and orchestrate co-simulation process. In our previous researches [21, 20], we developed a platform to manage co-simulation between AMESim [40], Flowmaster [34], Saber [8], Matlab/Simulink [26] and LMS Motion [27].
- **Co-simulation between hardware and software:** Hardware and software communicate with each other in order to implement real-time simulation [19, 14].
- **Co-simulation between software:** Different simulation tools use their tool adapters to communicate with each other during run-time co-simulation. The authors developed local/remote tool adapters for each tool to support run-time communication between them during co-simulation execution [1, 18, 11, 10].

Based on the findings from the literature review, we categorize tool integration into three types: (A) Control integration (B) Data integration (C) Co-simulation integration. In the following section we discuss suitable abstractions for these types of integration.

2.4 High-level Abstractions in Control integration

Control integration refers to tool operations which allow tool-chain users and tools some element of control including notifying users, activating tools, calling specific tool operations in other tools and triggering co-simulation and data exchanges or transformations. We make use of *technical space* for control integration referring to a common representation of tool operations, which allows unified access. *Notification* refers to tool operations whereby tools or users can communicate, for example sending an e-mail. *Activation* refers to tool operations that launch a tool with inputs and models. *Calling functionalities* refers to those tool operations where some function(s) of a tool can be triggered. *Trigger co-simulation* refers to tool operations where stakeholders or tools trigger co-simulation execution (e.g., system engineers press the button to start co-simulation). Generally, data exchange is needed between tools, but some tools cannot support data exchange directly. Therefore a transformation is needed to change the format so that tools can assess the data. *Trigger data exchange or transformations* can be used to trigger both data exchange and transformations.

In this paper, *control integration* refers to one type of tool-interactions from the dependency view. During control integration, a *tool* has one or more *tool adapters* used for control integration. *Technical space for control integration* refers to *Technical Space* to unify tool-operation allocation formats.

2.5 High-level Abstractions in Data integration

As presented in [6], there are two purposes of data integration: (a) Dealing with different *concrete syntax* [24] in the data model; and (b) Unifying *abstract syntax* and semantics, in different data models. Therefore, it could be said that three elements are needed: (1) A least two *tools* with one or more *tool adapters* used for data integration; (2) Tool interactions, data exchange directly between tools and data transformation. (3) A common representation of data referring to *TechniqueSpace* from tool-integration views.

2.6 High-level Abstractions in Co-simulation

Compared to previous research in [6], we extend new high-level abstractions in co-simulation integration to describe co-simulation features. Co-simulation allows *tools* to communicate with each other by using *tool adapters* as specific interfaces. During co-simulation, tool adapters execute data exchange between themselves and their host tools at each communication time step point. In Fig. 1, there are three types of API operations: (a) *Development API*; (B) *Configuration*

Fig. 1. API types during co-simulation

API; and (C) *Runtime API*. Before co-simulation execution, *Development API* is used for developing models to support co-simulation. *Development API1* is used to create a co-simulation model, for example, linking interface to blocks. *Development API2* is used to insert an interface for co-simulation in models. *Development API3* is used to generate execution files for co-simulation interface. *Development API4* is used to transform models into a common representation of co-simulation models, *technical space for co-simulation*, which allows unified access of exchange model format of co-simulation. *Configuration API* is used to configure tools for co-simulation, e.g., setting communication time step and setting solver types.

In general, co-simulation involves communication between executing softwares or between software and hardware and communication between executing softwares or between software and hardware by *technical space for co-simulation*, such as FMUs [25]. During co-simulation, *Runtime API1* is used for communication between co-simulation interfaces which is generated by adapters or FMUs (Functional Mockup Unit generated based on FMI) to exchange data between different models, e.g. send and get data from interface block. *Runtime API2* is used to control models or FMUs, e.g. initialize or terminate.

Abstraction proposal: As with co-simulation, we propose that three elements are needed: (1) At least one *tool* with one or more *tool adapters* which support co-simulation. During co-simulation, tools include *masters* and *slaves* [25]. (2) Co-simulation either directly between tools, between hardware and software or through *technical space for co-simulation*. (3) A common representation of model referring to *Technique Space* from tool-integration views.

3 Extending TIL To Describe Co-simulation Tool-chains

We extend TIL [5] to formalize co-simulation tool-chains in order to be able to describe them using a model-based approach. TIL is a domain specific language to describe tool integration solutions and thus is limited to describing

co-simulation features in a tool integration scenario. Therefore we adopt additional meta models to describe the co-simulation features in order to use them in describing the structure of a co-simulation tool-chain.

3.1 Extend TIL to formalize co-simulation tool-chains

The first step is to find the common patterns of co-simulation tool-chains' features and design the language concepts. Based on TIL, our extended language consists of two basic abstract types: components and connectors. We define components as: **User**, **Tool**, **Hardware**, **Tool Adapters**, **Repositories**, **Sequences** and **CosimControl**. Connectors are used to link components which are always directed. Compared with the original TIL, we added new meta models: (a) Co-simuControl, (b) Hardware, and (c) Co-simulation Channel. The concepts are introduced in detail below:

- **Users** - indicate stakeholders who have the interactions of tool-chains. Incoming **Control Channels** refer to a notification of users, e.g. by e-mail or notices in system messages. Outcoming **Control channels** refer to actions or tool operations triggered by the *Users*.
- **Tools** - refers to components that expose both functionalities and data of software or tools. **Tools** provide functional services to tools and hardware exposing tool functionalities, data exchange and co-simulation by the use of **Tool adapters**.
- **Hardware** - refer to components that expose hardware used in co-simulation tool-chains.
- **Tool Adapters** - refer to components used to integrate tool functionalities and data or implement co-simulation. Capable of transforming, receiving data from tools or implementing co-simulation, e.g. code generation and co-simulation interface, it consists of two interfaces and the logic used to translate or communicate between them: (a) *Tool Interface* used to interact or communicate with the tool; and (b) *Integration Interface* for interacting with or communicating other integrated tools, integration platform or integration framework. In short, it could provide services that expose the functionalities of tools. For example, COM interface [3] is provided by Flowmaster to control Flowmaster execution [20].
- **Repositories** - refer to components that provide storage of tool data. Repositories accept data and model in specific formats. The inputs and outputs of **Repositories** are **Data Channels** referring to data or models import and export.
- **Sequencers** - refer to components executing multiple functional services of tools and tool adapters. They have incoming and outcoming **Control channels**. The incoming **Control channels**, - originating from any user or component - refer to the triggers of execution of sequences. The outcoming **Control channels** can activate a **Data channel** or components in a predefined order.

- **CosimControls** - refer to components mastering and controlling co-simulation of tools. They have incoming **Control channels**, **Data channels** and outcoming **Co-simulation channels**. The incoming **Control channels**, - originating from any user or component-, refer to the triggers of executions of co-simulations. The incoming **Data channels** represent data input. The outcoming **Co-simulation Channels** can present run-time communication with other tools or tool adapters during co-simulation. **Sequences** can control and trigger **CosimControls** to execute co-simulations.
- **Data Channels** - refer to directed connections transferring data between components to deal with the heterogeneous source components and target components and data transmission without transformation. They connect source components to target components. There are three types of data channels: (a) Explicit transformation, attached with a transformation to deal with structural heterogeneities between tool metamodels. (b) Implicit transformation, the data needed by target components is different to source components, e.g. the Data Channels between Meta-Edit and Matlab/Simulink in [22] (Introduced in Case Study 2); and (c) Data transmission without transformation, for example, users use a tool to open a source model in repositories or copy the model to repositories.
- **Control Channels** -refer to directed edges between a source component and a target component or channel. The source component activates the target components or invokes a functional service of the target components. For instance, if the **Control channel** points at a **Data channel** or a **Co-simulation channel**, it means the **Data channel** or the **Co-simulation channel** is activated.
- **Trace Channels** -describe traceability links of data between different components.
- **Co-simulation Channels** -refer to real-time data exchange between components. For example, **Co-simulation channel** could connect tool to hardware which means they communicate with each other during co-simulation. The co-simulation channels need to be linked to **CosimControls**. The **Co-simulation channel** can be activated by **Control channels**.

3.2 Concrete Syntax of extended TIL

In this paper, TIL is implemented in MetaEdit+ ⁴. OPRR concepts (Object, Point, Relationship and Role) represent a method used in MetaEdit+ to implement a domain specific model. The concrete syntax for describing co-simulation tool-chains is shown in Fig. 2. The extended TIL includes the components and connectors in Table 2. Currently, it is used to formalize the structure of the tool-chains. We use two existing industrial co-simulation platform case studies to illustrate the potential of the extended TIL.

⁴ We make use of MetaEdit+ to proof the concept of TIL and one of the further work to select a tool to support TIL

Fig. 2. Concept model of TIL

4 Case study

4.1 Case Study 1

In [21], we proposed a co-simulation tool-chain used for mechatronics system design. This project was implemented in 2010, and aimed to integrate subsystem models into a unified platform for integrated system design. In the customer's department, they had different organizational groups for subsystem design. In each design group, they had their own simulation tools: (A) Fuel system designers used Flowmaster to simulate dynamics behaviors of fuel system. (B) Hydraulic system designers used AMESim to simulate dynamic behaviors of hydraulic systems. (C) Control system designers used Simulink to analyze control systems. (D) Electronics designers used Saber and Simplorer to analyze the electrical and electronic systems. This isolation of tools meant that an overall analysis of system performance could not be implemented. We designed a platform for our customers and its architecture is represented using extended TIL in Fig. 3.

As Fig. 3 shows, the various developers (hydraulic system, electronics, control system, fuel system and electrical) use a *ModelManagementSystem* to upload their models to *repositories*. Then the system engineers use the *ModelManagementSystem* to download the models and apply a home-made *co-simulation control GUI* (developed based on Matlab/GUI) to insert the co-simulation interfaces. At short, the system engineers use *co-simulation control GUI* to trigger Matlab/Simulink to execute simulations.

4.2 Case Study 2

In [22], we devised a modeling environment for domain specific models to support co-simulation between Simulink, Carmaker and FMUs generated from Modelica models. It was used to analyze an autobraking system through simulations in

Table 2. Abstract syntax in extended TIL

Meta Model in MetaEdit+	OPRR concepts[17]	Description
<i>User</i>	object	refers to stakeholders who use tools
<i>Tool</i>	object	refers to tools used in tool-chains
<i>Tooladapter</i>	object	refers to tool adapters used in tool-chains. (It will be used in the future work).
<i>Sequencer</i>	object	refers to sequencers used in tool-chains
<i>Respositories</i>	object	refers to respositories used in tool-chains
<i>CosimControl</i>	object	refers to cosimcontrol used in co-simulations
<i>Hardware</i>	object	refers to hardware used in co-simulations
<i>DataChannel</i>	object	refers to data channel controlled by a control channel
<i>TraceChannel</i>	object	refers to trace channel controlled by a control channel
<i>Co – SimulationChannel</i>	object	refers to co-simulation channel controlled by a control channel
<i>ControlChannel</i>	object	refers to control channel controlled by a control channel
<i>Data</i>	Relationship	refers to data channel
<i>Trace</i>	Relationship	refers to trace channel
<i>CoSim</i>	Relationship	refers to co-simulation channel
<i>Cont</i>	Relationship	refers to control channel

Simulink, co-simulations between Simulink and Carmaker and co-simulations between Simulink, Carmaker and FMUs. Domain specific models were built in MetaEdit+.

Fig. 4 represents the tool-chain’s architecture. Mechanical engineers use MWorks [7] to build Modelica models for mechanical patterns of an autobraking system. Then they transform the Modelica models to FMUs and update them to *repositories*. System engineers build domain specific models in MetaEdit+, Simulink model library and Carmaker. They upload these models to *repositories* and transform the domain specific models to M language - used for Simulink operation, interface generation, Simulink model generation and Simulink execution. The co-simulation is triggered by the M language and executed by Simulink.

5 Evaluation and Future work

In this paper, we have represented the need for MBSE to design co-simulation tool-chain solutions. We analyze the possibilities to formalize the co-simulation’s architectures. We extended TIL, a high level domain specific modeling language to describe structures of co-simulation tool-chains. The industrial case studies illustrate how the TIL works. From the case study analysis, we summarize that:

- ☐ Systems engineering approaches could help stakeholders to capture the viewpoints and views of co-simulation tool-chains during design of the tool-chains’ architectures.
- ☐ Model-based representation could provide visual architectures of co-simulation tool-chains in order to enhance the stakeholders’ understanding and communication of tool-chain development.
- ☐ The model-based approach could be used as a potential extended method to achieve design automation for MBSE tool-chains.

Fig. 3. Using TIL to Represent the Architecture of Integrated Co-simulation Management System of Mechatronics

Fig. 4. Using TIL to Represent the Architecture of Integrated Co-simulation Management System of Autobreaking System

In our methodology, we have used domain specific models (extended TIL) to represent tool-chain architecture. The conceptual model is now in its first version and is a work in progress. In the future, we plan to adopt IEEE 1471 to formalize the functional requirements of MBSE tool-chains firstly. Then dependencies between stakeholders and MBSE tool-chains, - presented in [16] as an initial plan -, would aim to concentrate on how views conformations work to confirm the functionalities of MBSE tool-chains by using extended TIL. At last, another main point of future work is to evaluate the proposed TIL and to investigate analysis and synthesis capabilities.

6 Conclusion

Our methodology allows for the description of tool-chain's architecture in current status and has potentials to formalize MBSE tool-chain development and support its functional synthesis. We still need to analyze where a functional synthesis is possible. On the other hand, we have adopt service oriented approaches to develop MBSE tool-chains to support co-simulation. We will investigate whether methodology proposed in this paper can support code generation for java-based tool adapters. In closing, we feel our methodology has real potential to increase the efficiency of co-simulation tool-chain development.

References

1. Al-Hammouri, A.T.: A comprehensive co-simulation platform for cyber-physical systems. *Computer Communications* 36(1), 8–19 (2012)
2. Alachew Mengist, Adrian Pop, A.A., Fritzson, P.: Traceability support in open-modelica using open services for lifecycle collaboration (oslc). In: *Modelica 2017: Proceedings of the 12th International Modelica Conference; 15th and 16th May. Citeseer* (2017)
3. Baier, T., Neuwirth, E.: Excel:: Com. *Computational Statistics* 22(1), 91–108 (2007)
4. Becker, D., Singh, R.K., Tell, S.G.: An engineering environment for hardware/software co-simulation. In: *Design Automation Conference, 1992. Proceedings., 29th ACM/IEEE. pp. 129–134. IEEE* (1992)
5. Biehl, M., El-Khoury, J., Loiret, F., Törngren, M.: A domain specific language for generating tool integration solutions. In: *In 4th Workshop on Model-Driven Tool & Process Integration (MDTPI2011) at the European Conference on Modelling Foundations and Applications (ECMFA 2011)(6 June 2011)* (2011)
6. Biehl, M., Sjöstedt, C.J., Törngren, M.: A modular tool integration approach: experiences from two case studies (2010)
7. Chen, X., Wei, Z.: A new modeling and simulation platform-mworks for electrical machine based on modelica. In: *Electrical Machines and Systems, 2008. ICEMS 2008. International Conference on. pp. 4065–4067. IEEE* (2008)
8. Chwirka, S.: Using the powerful saber simulator for simulation, modeling, and analysis of power systems, circuits, and devices. In: *Computers in Power Electronics, 2000. COMPEL 2000. The 7th Workshop on. pp. 172–176. IEEE* (2000)

9. Dietz, S., Hippmann, G., Schupp, G.: Interaction of vehicles and flexible tracks by co-simulation of multibody vehicle systems and finite element track models. *Vehicle system dynamics* 37(sup1), 372–384 (2002)
10. Eker, J., Janneck, J.W., Lee, E.A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y.: Taming heterogeneity-the ptolemy approach. *Proceedings of the IEEE* 91(1), 127–144 (2003)
11. Fitzgerald, J., Larsen, P.G., Pierce, K., Verhoef, M., Wolff, S.: Collaborative modelling and co-simulation in the development of dependable embedded systems. In: *International Conference on Integrated Formal Methods*. pp. 12–26. Springer (2010)
12. Fitzgerald, J., Larsen, P.G., Verhoef, M.: From embedded to cyber-physical systems: Challenges and future directions. In: *Collaborative Design for Embedded Systems*, pp. 293–303. Springer (2014)
13. Gomes, C., Thule, C., Broman, D., Larsen, P.G., Vangheluwe, H.: Co-simulation: State of the art (feb 2017), <http://arxiv.org/abs/1702.00686>
14. Hoffman, A., Kogel, T., Meyr, H.: A framework for fast hardware-software co-simulation. In: *Proceedings of the conference on Design, automation and test in Europe*. pp. 760–765. IEEE Press (2001)
15. ISO/IEC, .: *Systems and software engineering - Recommended practice for architectural description of software-intensive systems*, vol. 2007 (2007)
16. Jinzhi Lu, De-Jiu Chen, D.G.M.T.: An investigation of functionalities of future tool-chain for aerospace industry. In: *INCOSE International Symposium*. Wiley Online Library (2017 in press)
17. Kelly, S., Lyytinen, K., Rossi, M.: Metaedit+ a fully configurable multi-user and multi-tool case and came environment. In: *Advanced Information Systems Engineering*. pp. 1–21. Springer (1996)
18. Li, S., He, L.: Co-simulation study of vehicle esp system based on adams and matlab. *Journal of Software* 6(5), 866–872 (2011)
19. Li, W., Joós, G., Bélanger, J.: Real-time simulation of a wind turbine generator coupled with a battery supercapacitor energy storage system. *IEEE Transactions on Industrial Electronics* 57(4), 1137–1145 (2010)
20. Lu, J.: Co-simulation for heterogeneous simulation system and application for aerospace. Master dissertation, Huazhong University of Science and Technology, Wuhan, China (2013)
21. Lu, J.z., Ding, J.w., Zhou, F.l., Gong, X.: Research of tool-coupling based electro-hydraulic system development method. In: *Proceedings of the 6th International Asia Conference on Industrial Engineering and Management Innovation*. pp. 213–224. Springer (2016)
22. Lu, J., Chen, D., Törngren, M., Loiret, F.: A model-driven and tool-integration framework for whole vehicle co-simulation environments. In: *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)* (2016)
23. Lynn, A., Smid, E., Eshraghi, M., Caldwell, N., Woody, D.: Modeling hydraulic regenerative hybrid vehicles using amesim and matlab/simulink. In: *Defense and Security*. pp. 24–40. International Society for Optics and Photonics (2005)
24. Mierlo, S.V., Tendeloo, Y.V., Meyers, B., Vangheluwe, H.: *The Handbook of Formal Methods in Human-Computer Interaction*. HumanComputer Interaction Series, Springer International Publishing, Cham (2017), <http://link.springer.com/10.1007/978-3-319-51838-1>
25. Modelica Association Project FMI: Functional Mock-up Interface for Model Exchange and Co-Simulation (07006), 1–120 (2013)

26. Ong, C.M.: Dynamic simulation of electric machinery: using MATLAB/SIMULINK. Prentice Hall (1998)
27. Ong, E.P., Spann, M.: Robust multiresolution computation of optical flow. In: Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on. vol. 4, pp. 1938–1941. IEEE (1996)
28. Qamar, A.: Model and dependency management in mechatronic design. Ph.D. thesis, KTH Royal Institute of Technology (2013)
29. Rowson, J.A.: Hardware/software co-simulation. In: Design Automation, 1994. 31st Conference on. pp. 439–440. IEEE (1994)
30. Standard, I.: INTERNATIONAL STANDARD ISO / IEC / IEEE Systems and software engineering engineering 2011 (2011)
31. Thomas, I., Nejme, B.A.: Definitions of tool integration for environments. IEEE software 9(2), 29–35 (1992)
32. Trappey, C.V., Trappey, A.J., Huang, C.J., Ku, C.: The design of a jade-based autonomous workflow management system for collaborative soc design. Expert Systems with Applications 36(2), 2659–2669 (2009)
33. Trčka, M., Hensen, J.L., Wetter, M.: Co-simulation for performance prediction of integrated building and hvac systems—an analysis of solution characteristics using a two-body system. Simulation Modelling Practice and Theory 18(7), 957–970 (2010)
34. Tu, Y., Lin, G.: Dynamic simulation of aircraft environmental control system based on flowmaster. Journal of Aircraft 48(6), 2031–2041 (2011)
35. Tu, Z., Zacharewicz, G., Chen, D.: Developing a web-enabled hla federate based on portico rti. In: Simulation Conference (WSC), Proceedings of the 2011 Winter. pp. 2289–2301. IEEE (2011)
36. Van Rompaey, K., Bolsens, I., De Man, H., Verkest, D.: Cowarea design environment for heterogenous hardware/software systems. In: Proceedings of the conference on European design automation. pp. 252–257. IEEE Computer Society Press (1996)
37. Wang, B., Baras, J.S.: Hybridsim: A modeling and co-simulation toolchain for cyber-physical systems. In: Proceedings of the 2013 IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications. pp. 33–40. IEEE Computer Society (2013)
38. Wasserman, A.I.: Tool integration in software engineering environments. In: Software Engineering Environments. pp. 137–149. Springer (1990)
39. Wetter, M.: Co-simulation of building energy and control systems with the building controls virtual test bed. Journal of Building Performance Simulation 4(3), 185–203 (2011)
40. You-Guan, Y., Guo-fang, G., Guo-Liang, H.: Simulation technique of amesim and its application in hydraulic system [j]. Hydraulics Pneumatics & Seals 3, 28–31 (2005)
41. Zitney, S.E.: Process/equipment co-simulation for design and analysis of advanced energy systems. Computers & chemical engineering 34(9), 1532–1542 (2010)