Int J Softw Tools Technol Transfer (2013) 15:363–374
DOI 10.1007/s10009-012-0230-5

REGULAR PAPER

A model-driven co-simulation environment for heterogeneous systems

Massimo Bombino · Patrizia Scandurra

Published online: 23 March 2012

Abstract Modern heterogeneous systems, due to their complexity and multifaceted nature, require flexible high-level design and simulation techniques that take into account both aspects of continuous time modeling and discrete event modeling. In this context, model-driven approaches and extensions of the OMG Unified Modeling Language for the Real-time Embedded system and System-on-Chip domains are gaining popularity, both in industry as well as in academy, since they offer a high degree of abstraction and provide a common framework for the design, simulation and configuration management. To establish advanced model-driven design environments for complex heterogeneous systems, possible strategies for combining such approaches and languages in a common modeling and simulation framework must be determined. This article proposes a model-driven continuous/discrete co-simulation framework based on the OMG SysML standard—a UML profile for system engineering applications—for discrete event modeling, and on the industry de-facto standard Matlab/Simulink for continuous time modeling. The proposed approach adopts a *code-in-the-loop co-simulation* schema where optimized C/C++ code—including glue code for time synchronization and model interaction—is automatically generated from Simulink and SysML models according to model-driven development principles. A supporting environment (also described here) provides simulation features such as remote graphical animation and model execution control.

M. Bombino
Atego company, Peschiera Borromeo, MI, Italy
e-mail: massimo.bombino@atego.com

P. Scandurra (✉)
Università degli Studi di Bergamo, DIIMM,
Viale Marconi 5, 24044 Dalmine, BG, Italy
e-mail: patrizia.scandurra@unibg.it

Keywords Real-time embedded systems · Model-driven engineering · Continue/discrete simulation · SysML · Simulink

1 Introduction

Heterogeneous systems, due to their complexity and multifaceted nature, require advanced engineering from different disciplines (such as software engineering, hardware–software co-design, and system modeling, to name a few) with flexible high-level design and simulation techniques that take into account both aspects of *continuous time modeling* and *discrete event modeling*.

During system design, often the core of the activity is the definition of a special algorithm or the *plant*. Consider, for example, the Flight Control System (FCS) of a flying vehicle (like a jet), or the Radio Transmission Algorithm (like W-CDMA) for a wireless communication system like UMTS or OFDMA for WiMAX. Most algorithms work in continuous time domain: the input values are taken, the control law is applied, and the output values are immediately made available to the actuators. The algorithm specialists often start to work before system engineers, since they have to simulate and validate in advance all the details of the algorithm with a separate tool. One of the most used tools for this purpose is Mathworks Matlab/Simulink. The Simulink model, possibly composed by sub-modules, is typically a continuous time-description of the algorithm; but, this model is only a small part of the overall system design. On the other hand, system-level events are usually asynchronous and discrete; consider, for example, user interactions, threshold passing, alarms, etc. The system state and its changes as reaction to certain discrete events are usually described in terms of a *state model* using a state-like formalism such as Finite State

Machines or extensions. Tool-supported co-design and simulation methodologies that suit the peculiarities of the two worlds (continuous and discrete) are, therefore, necessary.

In this context, *Model-driven Engineering* [14] approaches to the design and development of software and systems are gaining popularity, both in industry as well as in academy, since they offer a high degree of abstraction and provide a common framework for the design, simulation and configuration management. These approaches rely on high-level modeling languages and automatic model transformations and code generation techniques to achieve significant boosts in both productivity and quality. A great variety of modeling languages for the real-time embedded system and System-on-Chip (SoC) domains have been emerging as extensions (or *profiles*) of the OMG Unified Modeling Language (UML)[18]—like the SysML (Systems Modeling Language) [16] for system engineering applications, the MARTE (Modeling and Analysis of Real-Time Embedded Systems) [10], and the UML for SoC [19] for System-on-Chip design, to name a few.

However, more synergies and integration are necessary before model-driven IDEs become available for use in multidisciplinary contexts. In particular, research is still undergoing in the continuous/discrete co-design and co-simulation area. The idea of co-simulation is to bring together the Continuous Time (CT) together with Discrete Events (DE), to provide a more effective environment for simulation covering several different aspects. Some difficulties in the development of such tools are (1) the heterogeneity of concepts and timing aspects manipulated by the discrete and continuous components, and (2) the communication and synchronization issues with respect to accuracy and performance constraints of continuous/discrete system model simulation.

To establish advanced model-driven design environments in such area, possible strategies for combining high-level modeling languages in a common design and simulation framework for continuous/discrete systems must be determined. Recently, the Matlab/Simulink framework and the OMG SysML and MARTE modeling languages are gaining increased attention for electronics system-level design [5,20]. While Matlab is commonly used to model signal processing intensive systems, UML-based notations have the potential to support innovative methodologies which tie the architecture, design and validation tasks in a unified manner and from a system engineer perspective.

Along this direction, this article proposes a model-driven co-simulation approach for continuous/discrete systems based on the OMG SysML standard (as implemented in the Atego Artisan Studio) for discrete event modeling, and on the industry de-facto standard Matlab Simulink for modeling algorithms and continuous time aspects of a system. On one hand, Simulink allows the system engineers to: model the mathematical constraints of a system, simulate the system, and refining algorithms and system parameters. On the other hand, SysML allows system engineers to: model the structure and behavior of a system with Block Diagrams and State Diagrams, model the mathematical constraints of a system with Constraint Blocks and Parametric Diagrams. These constraints/parameters may be synchronized with their corresponding counterpart within Simulink, expanded on, used for simulation, refined as necessary, and reversed back at SysML diagram level for model animation. In particular, a *code-in-the-loop* CT/DE co-simulation schema is here proposed. Indeed, co-simulation is based on a target implementation language (C/C++ programming languages) that is used as common execution language for both Simulink and SysML models.

The proposed methodology is supported by an environment that relies on model-driven development principles to allow C/C++ code generation from Simulink and SysML models. It exploits the native code generation capabilities of two existing selected tools, Matlab Simulink and Artisan Studio, and model-to-model transformations of the second one, to generate source code for the DE and CT simulations and additional "glue code" (including scripts and project/make files) to allow the exchange of events and data between the CT/DE simulators and resolve time synchronization aspects. The resulting code is automatically compiled for a specific target (a host PC or an embedded target) and runs at real-time or scaled real-time. The tool provides also simulation features such as remote graphical animation and model execution control.

With respect to the state of the art (as better explained in Sect. 2), there are two innovative aspects in our contribution. First, the use of a model-driven approach and, in particular, of the SysML standard improves significantly the design of systems and of embedded software. One of the advantages of combining SysML with Simulink compared to classical Simulink/Stateflow or Simulink/UML solutions is that SysML has a more precise semantics and offers better support to tie in an unified perspective the specification, analysis, design, verification, validation, requirements traceability, and documentation of system engineering. These systems may include hardware, software, information, processes, personnel, and facilities.

Second, the adoption of a co-simulation schema based on (C++) code, that is generated automatically from SysML and Simulink models and executed natively on a target platform, allows faster simulations and better performance than co-simulation approaches where Simulink and the UML tool communicate with each other via a coupling tool [20].

This article is organized as follows. Section 2 compares this work with existing approaches in the literature. A brief description of the SysML language is given in Sect. 3, while the reader is assumed to be familiar with Simulink. The CT/DE co-simulation methodology is presented in Sect. 4.

Section 5 describes the co-simulation environment supporting the proposed methodology. Section 6 provides an illustrative case study of a discrete/continuous application. Finally, Sect. 7 concludes the article and outlines some future research directions.

2 Related work

Experiments to integrate the Matlab Simulink tool, commonly used to model signal processing intensive systems, with UML/SysML simulation tools for co-simulation purposes already exist. As stated in [20], two different approaches for coupling UML and Simulink have been proposed so far: (1) *model-in-the-loop* co-simulation via an intermediate coupling bus that allows the two simulations (Simulink and UML simulations) to communicate, and (2) *code-in-the-loop* co-simulation based on a target implementation language (usually C++ or the Matlab code itself) that is used as common execution language for both models. Both approaches vary in the provided abstraction and effective integration. The first co-simulation approach requires special attention to the synchronization between the UML tool and Simulink. Both simulations exchange signals and may run concurrently, in the case of duplex synchronization, or alternatively in the sequential case. The former solution increases the simulation speed, whereas the time precision of the exchanged signals is higher in the latter case [20]. The second co-simulation approach requires specific development frameworks which ease the creation or the generation of the executable code from UML and Simulink, and their connection; but it allows faster simulations than the first approach since simulations and communications are at code level. As an example of the first co-simulation approach, the Exite ACE tool from Extessy [8] allows the coupling of a Simulink model with Artisan Studio. The second approach is adopted for example in the Constellation framework [9] and in the General Store integration platform [7]. Both tools provide a unified representation of the system at UML model level on top of a code implementation level. The Simulink subsystem appears in Constellation as a component which can be opened in Matlab, whereas a UML representation of the Simulink subsystem is available in GeneralStore, based on precise bidirectional transformation rules.

The work proposed in this article is an example of the second co-simulation approach, as the integration is effectively made at (C++) code-level. Specifically, the proposed methodology: (1) offers an easy code-in-the-loop synchronization schema for continuous/discrete co-simulation to minimize interactions between simulators; (2) generates the synchronization "glue code" in an automatic way from input models by a model-driven approach that relies on a small set of specific *stereotypes* (extensions/annotations of SysML modeling concepts) and transformation rules applied to the input SysML model; (3) supports the native running on a target embedded system (no instances of the Simulink and SysML Artisan Studio tools are required on the target); (4) provides remote model debugging/animation features (and in particular, the real-time animation of the SysML state machine diagrams by the data coming continuously from the simulation).

The work in [3] is similar to ours. Their co-simulation approach relies on Simulink for the continuous simulation and on SystemC[1] for the discrete simulation. However, they do not rely on a MDE approach. In our code-in-the-loop co-simulation schema, optimized C/C++ code is automatically generated, instead, from Simulink and SysML models according to model-driven development principles. Our approach does not support the generation of SystemC code merely to allow hardware–software co-simulation. SystemC is a code-based formalism too specific to the Soc domain and as modeling language may not cover all aspects required in a multidisciplinary domain to describe complex heterogeneous systems as the SysML (and therefore C/C++) may do.

The Ptolemy project[2] studies modeling and simulation of concurrent, real-time, embedded systems. The focus is on the use of well-defined (and possibly heterogeneous) models of computation that govern the interaction between components. However, the Ptolemy framework does not rely on standard notations, while our framework is completely SysML-based (OMG standard) and Simulink-based (de-facto standard).

The work in [4] is also similar to our approach. The authors propose the use of the UML as a single modeling language for initial specification, and the synthesis of a Simulink model from the UML model. Thought using the UML as it is provides the advantage of using a standard language that is widely accepted in the software engineering community, their work would benefit from the use of SysML (instead of UML).

In [15], an approach for transforming Simulink models into UML composite structure diagrams (for the structural view) and activity diagrams (for the behavioral view) is presented. The work has been carried out in the context of the ATESST project [2] in the automotive domain. In this context, an architecture description language, named EAST-ADL2, for automotive embedded systems has been defined as an UML profile. Though their work is too specific to a traditional automotive software engineering process, it would be worth in the future to study the feasibility of a possi-

[1] SystemC [17] is an open standard in the EDA (Electronic Design Automation) industry. Built as C++ library, SystemC is a programming language providing abstractions for the description and simulation of SoCs.

[2] http://ptolemy.eecs.berkeley.edu/.

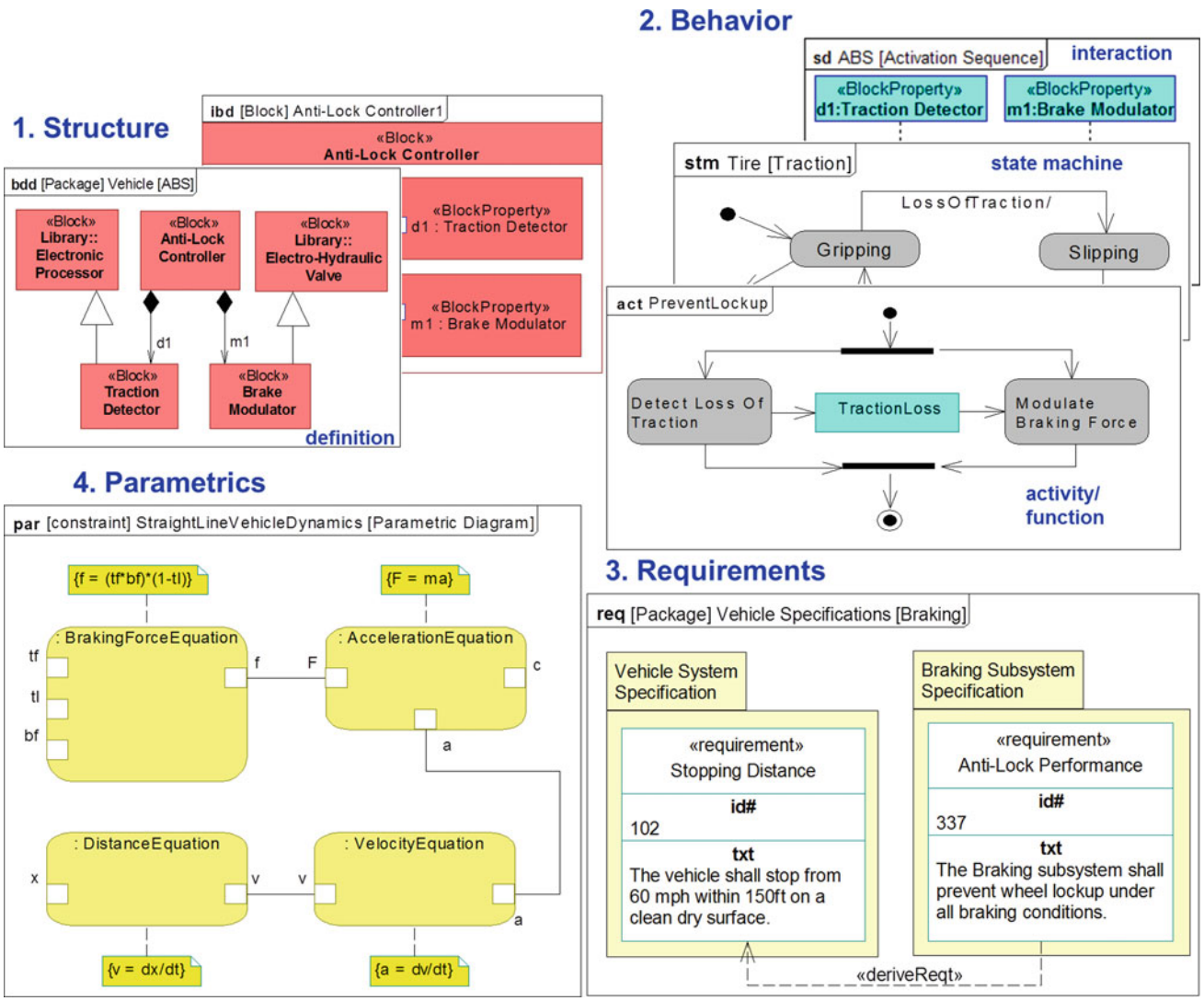

Fig. 1 The four pillars of SysML—adapted from [16]

ble integration with our approach that relies, instead, on the SysML state machine diagrams for modeling the discrete system behavior and on parametric diagrams for modeling constraints of the system's parameters. Another similar work that still relies on the UML activity diagrams is proposed in [15].

3 The SysML modeling language

SysML is a general-purpose modeling language for system engineering applications (including Aerospace and Defense, Automotive, and IT system applications) providing support for the representation and simulation of heterogeneous behavior [11]. A SysML model provides, therefore, a graphical representation of the system being developed, enabling a design team to share ideas and to cope with issues early thus preventing problems that would otherwise delay development and degrade design quality.

SysML is defined as an extension of a subset of the UML using the UML's *profile mechanism* [18]. SysML does not adopt all UML diagram types (object diagram, communication diagram, interaction overview diagram, timing diagram, and deployment diagram are not included, for example), as SysML is based on a minimal subset of the UML. Figure 1 shows the four pillars SysML diagrams (slightly adapted from [16]).

For *requirement traceability*, SysML introduces a *requirement diagram* to structure the requirements and link these to the system architecture and test procedures.

For *structural modeling*, SysML offers the *Block Definition diagram* (BDD) and the *Internal Block diagram* (IBD) to model the collaborative and hierarchical structure of a system in terms of modular units called *blocks*. Blocks may include both structural and behavioral features, such as properties and operations, to represent the state of the system and

behavior that the system may exhibit. A block can include properties to specify its values, parts, standard and flow ports, and references to other blocks.

Parametric diagram (ParD) is another new structural diagram. The parametric diagram is used to model system parameters and relate them to each other. SysML allows also the specification of *constraints blocks* to represent mathematical expressions constraining the physical properties of a system. Such constraints can also be used to identify critical performance parameters and their relationships to other parameters, which can be tracked throughout the system life cycle.

For *behavioral modeling*, SysML offers activity diagrams, sequence diagrams, and state machine diagrams (SDs), slightly improved and simplified from the UML2 version.

4 Co-simulation methodology

In the co-design and co-simulation process proposed here, it is assumed that: on one hand, the algorithm specialists provide and validate a Simulink continuous-time model of the plant or the main algorithm, and on the other hand, system engineers provide a SysML model of the entire system including requirements diagrams for requirement traceability, BDDs and IBDs describing the collaborative and hierarchical structure of the system in terms of blocks, ParDs describing the constraints and parameters of system's blocks, and SDs modeling reactive discrete behavior. SysML SDs are crucial for a code-in-the-loop co-simulation schema (like the on proposed here), because they provide a well-consolidated formal description of the states (and sub-states) changes of a system.[3] Moreover, SDs can be easily animated at code-level since their synthesis into code is easily performed by exploiting well-known generation patterns (*Run-To-Completion*, *state pattern*, *stable table pattern*, etc.).

Usually, these two Simulink-SysML worlds remain separate. Technically, on one hand, Simulink has a native GUI for host-based simulation, supporting graphical plot of the output variables with scopes. The Simulink environment is also endowed of a component, called Real Time Workshop (RTW), that allows C/C++ code generation for a generic or specific target, so the simulation could be run natively without GUI support. On the other hand, Artisan Studio (the tool adopted here for modeling in SysML) supports—by the Automatic Code Synchronizer (ACS)—code generation from SysML state diagrams, and it has also a simulation environment for state machines. It also includes a graphical debugger with basic animation capabilities for state

[3] SysML SDs are formal enough to be used for this purpose and, in contrast to the Simulink/Stateflow formalism, are well integrated in the overall SysML-based system design activity. Other SysML behavioral diagrams, such as Activity Diagrams, are not yet formal enough for simulation purposes.

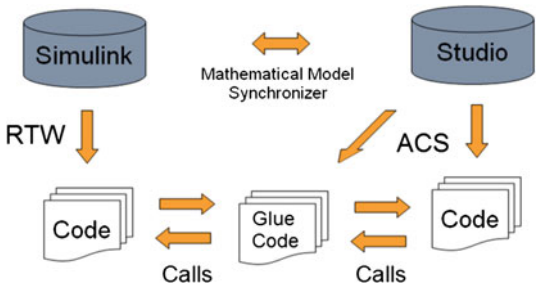

Fig. 2 Code-in-the-loop co-simulation schema

diagrams and the opportunity to interactively send/receive events to/from the simulator.

The Simulink experts and SysML engineers have no easy way to exchange parts of the two models, even when the structure and the modeled elements correspond to the same concepts. The Simulink simulation may be self-contained, but usually requires some inputs and outputs coming from the remaining parts of the system. Data are usually provided in form of vectors or stored signals, and the output is recorded in form of vectors or graphs. The basic idea proposed here for synchronizing the two worlds consists in aligning the two Simulink-SysML models for both the *structural* and the *behavioral* views.

For the structural view, only the topological structure of the two models is synchronized, while inner details of the algorithm (such as descriptions of integrators, PIDs, sums, etc.) may not be expressed in SysML and remain completely described in the Simulink model. The synchronization process assures that the Simulink model and SysML model are constantly kept aligned. With the Artisan Studio, this is feasible in practice through a component named Mathematical Model Synchronizer (MMS); therefore, the proposed framework (as better described in the Sect. 5) exploits such a component to allow an automatic and easy alignment between Simulink modules from one side and SysML blocks with associated parameters (IBDs and PDs) on the other side.

A tighter synchronization is achieved at behavioral level (behavioral view), by combining and simulating together CT aspects described in the Simulink model with the DE modeling provided in terms of SysML SDs. Up to now, most of the approaches (to the best of our knowledge) are based on a direct connection at run-time (through an appropriate coupling tool) of the tools' GUIs. The proposed approach, instead, adopts the code-in-the-loop co-simulation schema shown in Fig. 2: by exploiting the code generation capabilities of both Simulink RTW and Artisan Studio ACS, an optimized code is automatically generated (including glue-artifacts) from both the Simulink and the SysML models and then compiled and executed directly on the target platform allowing, therefore, real-time simulation natively on the target. Below, more details on the proposed co-simulation schema are provided.

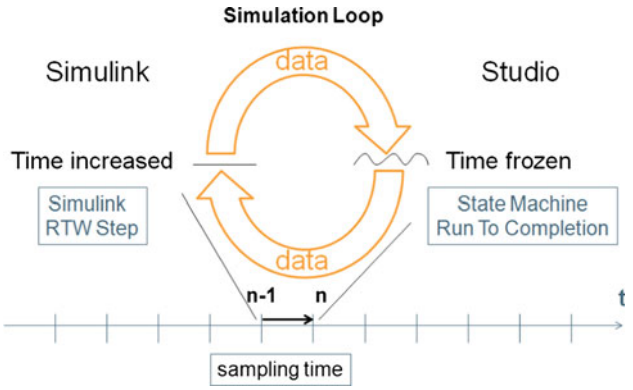

Fig. 3 Co-simulation loop

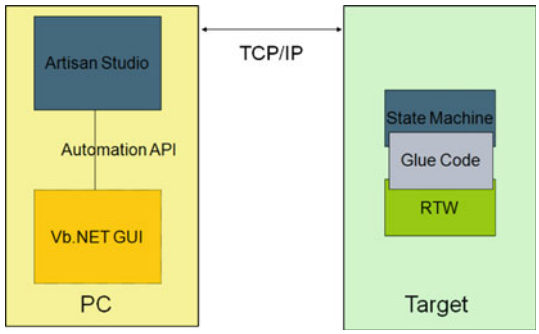

Fig. 4 SysML/Simulink co-simulation environment architecture

4.1 Time synchronization

Ensuring a consistent notion of time is crucial to guarantee proper synchronization between the two timelines: the Simulink (continuous timeline) and the SysML simulation (discrete timeline).

Simulink is based on continuous time model, but the inner algorithms require to work on a minimal step size that should be chosen depending on the frequency of the signals, according to the *Nyquist Theorem*. So a simulation *sample time* or *step size* should be selected by the user (or the default one accepted): this is the basic unit for calculation of all the algorithms and, therefore, for all the co-simulation aspects. The basic simulation schema consists of a tight loop where, among other things, the timing aspects are evaluated, and when the simulation timing step is expired, the main core function of the algorithm, step *rtOneStep*, is invoked.

SysML SDs simulation, as implemented by the ACS code synchronizer, is based on the *RunToCompletion* semantics defined in the UML specification [18]: every time an event is injected into the system (*event occurrence processing*), a transition is taken, a timer elapses, and the *RunToCompletion* step is executed.

The proposed co-simulation schema is essentially an extension of the basic Simulink simulation loop, where normally only the *rtOneStep* is invoked, as follows:

1. data input values from Studio to Simulink are transferred
2. an *rtOneStep* is executed
3. data output values from Simulink to Studio are transferred
4. a *RunToCompletion* step is executed to evaluate change conditions at SysML state machine level
5. optionally, internal events are generated for graphically animating the SysML state diagram.

The resulting simulation loop is depicted in Fig. 3. In this way, Simulink plays the role of *master* of the co-simulation schema, and at every simulation step, it computes through the *rtOneStep* routine the new values of the simulation from the input values. Once the output values are exchanged between the two worlds, a call to the *RunToCompletion* routine then verifies the change conditions and triggers events in the SysML SD simulation (the *slave*). The simulation normally is real-time, running at full speed; it may be slow down, *scaled-realtime*, by allowing a user-defined simulation delay be elapsed at every simulation step. Note that during the *RunToCompletion* step, the CT simulation is "frozen" (see Fig. 3), until the call to the *RunToCompletion* is terminated.[4]

4.2 Discrete event generators: timers and user events

Events in SysML SDs can be normal events (call events, signal events, and change events) or time events. Normal events can be injected from the user simulation interface itself, or caused by other internal actions of the system, or externally in the Host PC with the Inject/Query Dialog and GUI interface, as better described in Sect. 5.6.

Time events are generated by timers, started at simulation startup or by other conditions, and handled by OS calls. When a timer expires, automatically a callback routine is called that in turn invokes the *RunToCompletion* step. In this initial approach, the timescale for time events is absolute,[5] so it is independent from the Simulink sampling time and from the simulation delay.

5 Co-simulation environment

Figure 4 shows the overall architecture of the proposed co-simulation environment in terms of components and communication bindings. Essentially, the simulation code can be generated from the Simulink and SysML models for a normal Windows PC or for a specific embedded target—currently

[4] The amount of processing of the *RunToCompletion* routine must be guaranteed to be not too big, to avoid to slow down the simulation.

[5] OS timers at SysML level could be replaced by simulation-time timers at code level, to remove the limitation that they actually are absolute.

supported OSs are Linux and VxWorks—and then compiled for the specific architecture. The compilation and simulation facilities are provided directly within the Artisan Studio (the SysML modeling tool) through a context-menu. Additional features are also supported such as graphical animation of the SysML SDs on the host PC, connected through a TCP/IP connection to the target performing the simulation, and the capability of sending/receiving events—*Inject/Query events*—to/from the simulation environment interactively through a graphical interface (for example a VB.NET GUI).

The main features of the co-simulation environment are detailed in the following subsections.

5.1 Simulation parameters and initialization

The co-simulation process usually requires some input parameters to start. In the proposed approach, these parameters are provided directly into the SysML description, as default values. The ACS tool takes into account these values when it generates the code for the simulation.

The co-simulation process requires an initialization for:

- *Resource allocation* memory, timers, algorithms, and other resources that should be allocated before the simulation starts.
- *Working variables* some working variables should be initialized for the simulation steps to be executed correctly.
- *Connection to remote Artisan Studio instances* the TCP/IP connection from the target code to the host Artisan Studio instance should be setup before the simulation starts, for activating the Remote Animation feature.
- *Basic simulation parameters* it regards the duration of basic simulation time steps and should be defined by the user or by default values. These basic simulation parameters include:
  - *Start Time* default is 0 s, it represents the initial time of the simulation
  - *Stop Time* default is 100 s, it represents the final time of the simulation
  - *Step Size* default is 0.1 s, it represents the finer step for the simulation
  - *Step Delay* default is 100 ms, it represents a delay inserted between two simulation steps to scale down the simulation realtime. 0 ms is for real-time simulation.

The source code of the program (a file `Main.Cpp`) executing this initialization is generated automatically by an ACS Generator of dynamic-link libraries (DLLs). This source file is a collection of different contributions: *GRT MAIN.CPP*, the base module usually provided with the Simulink RTW component and contains all the initialization and allocation aspects related to the continuous time algorithm; *State Machines setup* done automatically from the context class constructor; CLI (Command Line Interface) to accept user-defined values for simulation parameters.

Table 1 Structural mapping between Simulink and SysML

Simulink element	SysML element
Model Reference	Constraint Property
Subsystem	Constraint Property
Port	Constraint Parameter [ParD]
Port	Flow Port [IBD]
Data stores	Block Properties

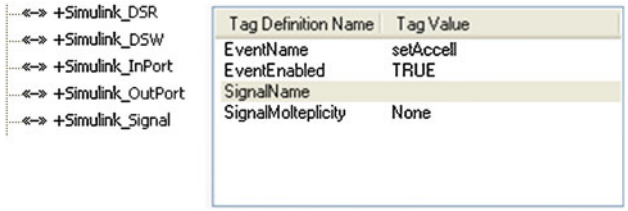

Fig. 5 SysML extension for CT/DE events and data exchange

5.2 Automatic structural synchronization

Structural synchronization between the Simulink and the SysML models is achieved through the automatic creation of a Simulink block diagram from a SysML Parametric Diagram, or vice versa—the *import/export functionality* of the Mathematical Model Synchronizer of Artisan Studio. The bidirectional mapping between structural Simulink and SysML modeling elements adopted for such goal is reported in Table 1. After a change to the structural view of either the Simulink or the SysML model, one can therefore immediately update its corresponding counterpart to reflect those changes.

5.3 Model annotation for events and data exchange

Another important aspect regards the events and data exchange between the CT view (the Simulink simulation) and the DE view (SysML SDs simulation).

In Simulink, different kind of signals and value ports are available for events and data exchange: *Data Store Read/Write*, *Ports (In, Out)*, and *Signals*. All these values could be typed by basic Simulink datatypes (like Uint8, Uint16, etc.). The corresponding counterpart in SysML are essentially blocks property values (i.e., attributes for a class) typed by data types that match the corresponding Simulink data types.

To support run-time synchronization between these concepts and allow events and data exchange, the SysML model is annotated through a small set of *stereotypes* (see Fig. 5) and their associated *tagged values* that have been defined accord-

ing to the standard UML extension mechanism of *profiles*.[6] Tagged values associated to these stereotypes provide the following information:

- *Event Enabled* if set to TRUE, it indicates that the relative *Event Name* event must be injected in the SysML SD every time the value changes—*Signal Name* name of the corresponding signal/port in the Simulink model (same name used in the SysML model by default)
- *Signal Multiplicity* multiplicity of the signal/port in the Simulink model (default is 1 for scalars).

The ACS tool takes into account these annotations during the code generation phase. It checks the consistency of the mapping between the Simulink and the SysML datatypes, and eventually it generates a warning.

Currently, events and data exchange is only one-way, i.e., the direction is either from the Simulink algorithm to SysML SD (involving Outport or DatastoreWrite) or from SysML SD to the Simulink algorithm (involving Import or DatastoreRead). No support for In-Out ports, i.e., bi-directional data exchange, is supported yet, as it requires a certain reasoning and design effort to avoid any ambiguity issue that may arise during simulation when the same value is modified simultaneously in both directions.

5.4 Automated script generation for compilation

The code provided by Simulink Real Time Workshop and the standard Artisan Studio ACS, usually may be compiled and run independently. Since the final result, together with the glue code for co-simulation, should run on the Host PC or on a target, an adequate makefile should be generated. Currently, the Window OS and Microsoft Visual Studio are supported,[7] so through a modified C++ Generator DLL, the automatic generation of a Microsoft Studio Visual C Project (VC Project) is provided. To this purpose, another stereotype, `Automatic_SM`, applied to the SysML SD diagram identifies the Simulink model to which the SysML model should be connected. This stereotype provides additional information for the generation of the Makefile/VC Project through the following tags:

- *Model Name* of the Simulink model to synchronize
- *Model Path* path where Simulink RTW generates files
- *Synchronized SM* a boolean indicating if the co-simulation is active
- *Simulation Log* a boolean indicating if a simulation log is available
- *Build Configuration* select Debug or Release.

5.5 Model-driven code generation

The code generation facilities already available in the Simulink environment and Artisan Studio are both exploited.

On one hand, the code generated by the Simulink RTW component usually is used "as it is", but it can be easily interfaced to other simulation tools to control the simulation and provide input and outputs in a certain manner (like in the proposed co-simulation approach).

On the other hand, the Artisan Studio has the ACS real-time synchronizer that keeps model and code synchronized according to the chosen Generator DLL. These DLLs range several different target languages and applications. The default code generation setting for SysML SDs was adopted for the proposed approach; namely the *RunToCompletion* code generation pattern and the generator DLL for standard C++.

The ACS synchronizer is also customizable, through the Artisan Studio Transformation Development Kit (TDK) for model transformations, allowing therefore the creation of new patterns for code generation and *model-to-model* transformations. The languages natively supported in TDK are SDL (Syntax Definition Language) and RSN (Reverse Syntax Notation). This feature has been exploited to customize the default code generation mechanism for SysML SDs to adapt it to a different target architecture, taking advantage of the OS features and APIs available on the target (e.g., thread synchronization and communication mechanisms like semaphores, mutex, locks, etc.). In particular, the SDL language of the ACS TDK has been used for implementing the transformation that generates the "glue code" to interface the SysML model with the Simulink model. This generated interface covers the synchronization aspects of the co-simulation presented previously, i.e., data types handling, initialization and model synchronization, generation of project/makefiles, etc.

5.6 Simulation logger and tools for simulation control

A simulation snapshot is shown in Fig. 10 for the case study presented in the next section. When the user starts the simulation, basically a console-based log is displayed for the duration of the simulation (see the left-lower corner of Fig. 10). The information displayed by the simulation logger regards the simulation time, exchanged signals and ports values, and system events. In addition, remote graphical animation for SysML SDs is supported within Artisan Studio,

[6] A UML profile is formally a set of stereotypes, each defining how the syntax and the semantics of a UML modeling construct is extended for a target application domain. A stereotype can define tagged values to enrich a modeling construct with further properties and can express constraints to enforce semantic restrictions.

[7] Other operating systems could be easily added: this aspect is taken into account also by the ACS component with the PowerMaker feature (allowing automatic generation of Makefiles according to basic rules).

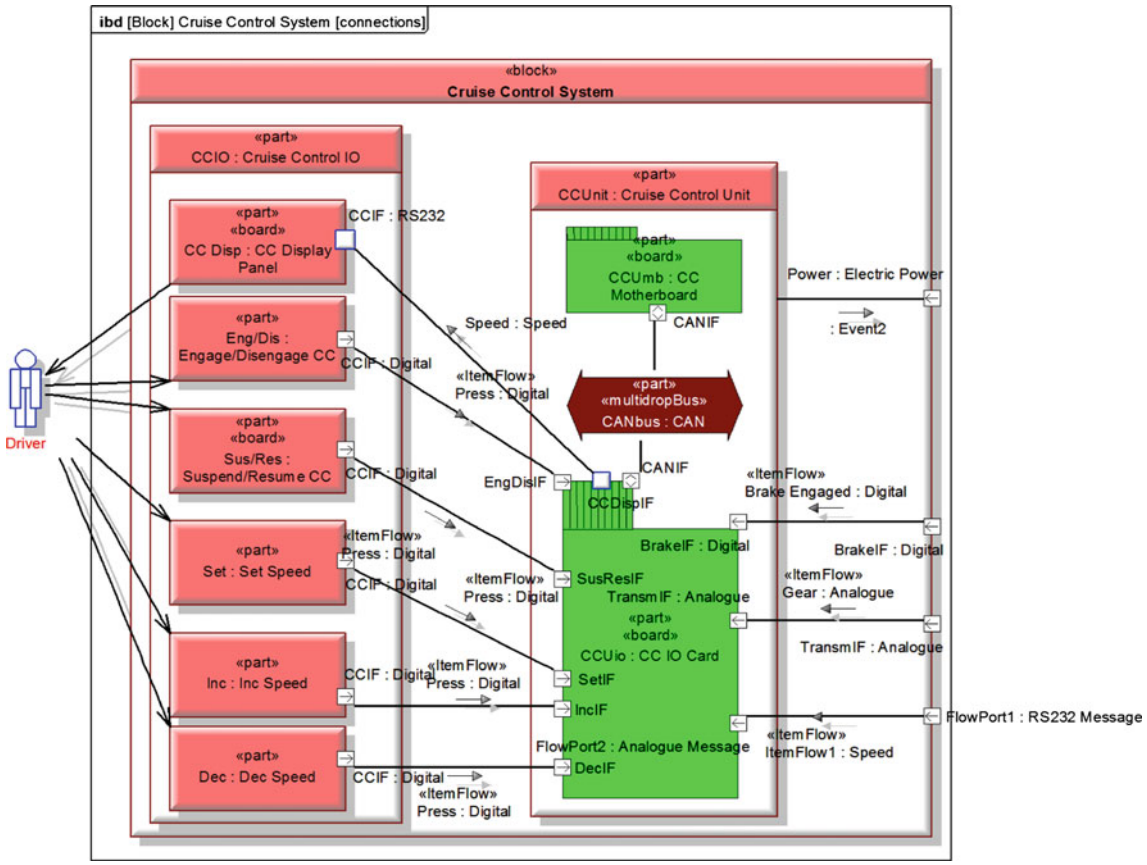

Fig. 6 SysML Internal Block diagram of the CCS vehicle

and also commands to interactively Inject/Query events during simulation.

5.6.1 Remote animation capabilities

The simulation could run natively on a target (of course it could be the same Host PC used for simulation) and could be connected through the TCP/IP protocol to an Artisan Studio instance running on the Host PC. Every event and transition in the State Machine is broadcasted to the remote Studio instance, and an Instance Diagram is created as shown in left-upper corner of Fig. 10. There is a color convention: red to denote the current state; blue to indicate the simulation control flow (i.e., the active states and triggered transitions), and black to denote the remaining parts not yet explored.

5.6.2 Inject/Query events features

The Artisan Studio GUI may be used also to inject user events during simulation. An "Inject/Query Events Dialog" (see the right-lower corner of Fig. 10) is used for this purpose. This window contains a dynamically generated list of system events and variables that the user could inject at any moment during the simulation. Every time an event is sent, the *RunToCompletion* step is automatically executed on the corresponding state machine.

5.6.3 Graphical interface

Artisan Studio supports OLE Automation to control all aspects of the simulation. This mechanism can be exploited to easily build a system-specific GUI with a rapid development tool such as Microsoft Visual Basic.NET. This GUI can then be used to interactively generate and inject events, and to display simulation values in real-time (see the right-upper corner in Fig. 10).

6 Case study

As complete case study, a cruise control system (CCS) for an Hybrid Sport Utility Vehicle (HSUV) is presented. The starting point was the CCS vehicle model in SysML (see some fragments of it in the SysML diagrams shown in Figs. 6, 7 and 8) as provided by the Artisan Studio tool and recommended by the SysML Specification [16]. In particular, a SysML SD (see Fig. 7) represents the basic operational states of the vehicle. The core CCS algorithm was implemented,

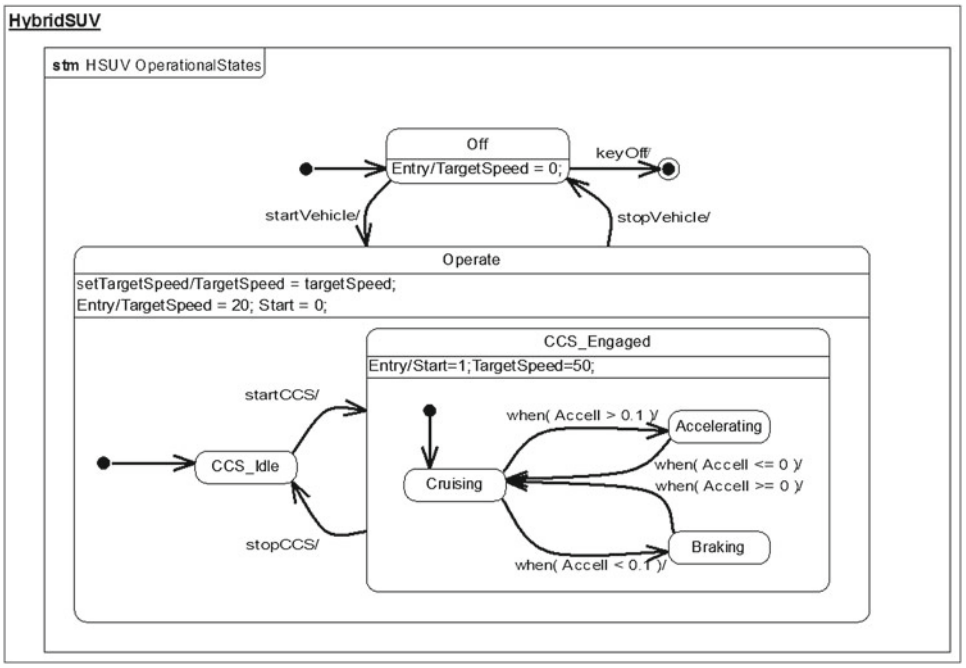

Fig. 7 SysML State Machine diagram of the CCS vehicle

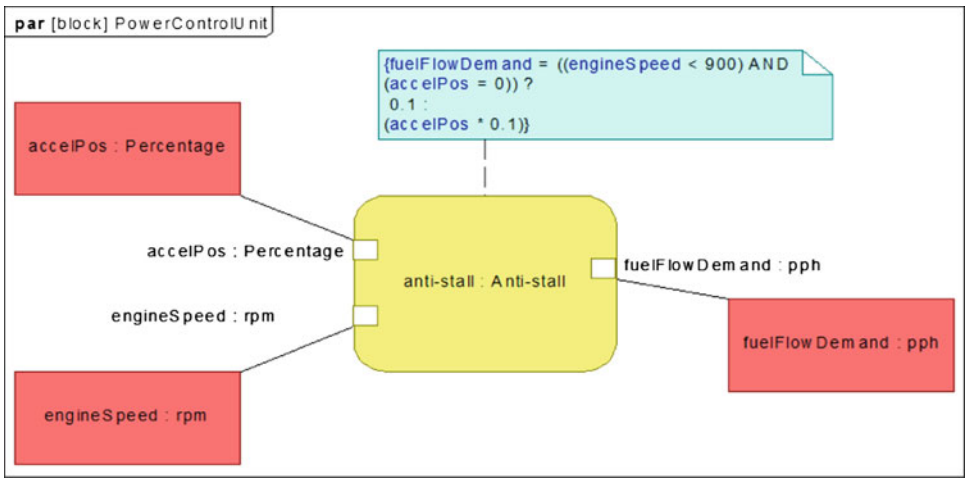

Fig. 8 SysML Parametric diagram of the CCS

instead, in Simulink (see Fig. 9 for a fragment of the Simulink block diagram).

From this base, few changes have been made to the original SysML model and to the underlying environment before co-simulation:

- The SysML SD has been annotated with the *SimulinkSM* stereotype to provide information about the Simulink model and RTW code;
- The SysML ParD (see Fig. 8), describing the constraints of the CCS algorithm, has been exported to Simulink with the Studio structural synchronizer;
- The blocks properties involved in the data exchange with Simulink (Target Speed, Start, Speed, Acceleration) have been annotated with adequate *SimulinkInport-Outport* or *SimulinkDSR-DSW* stereotypes;
- Change events on the transitions between states have been created to detect changes in the values calculated by Simulink;
- Artisan Studio ACS has been started with the Co-simulation Generation DLL and the code and makefile have been generated;
- The final code has been compiled for Windows and an executable has been created;
- Finally, a graphical interface for exchanging events has been created with VB.Net.

The Simulink implementation of the CCS algorithm was then co-simulated with the HSUV SysML state diagram (see Fig. 7). Once the vehicle is started and the CCS is engaged, the state of the machine (ranging in the set `{Cruising, Accelerating, Braking}`) is no more static or forced

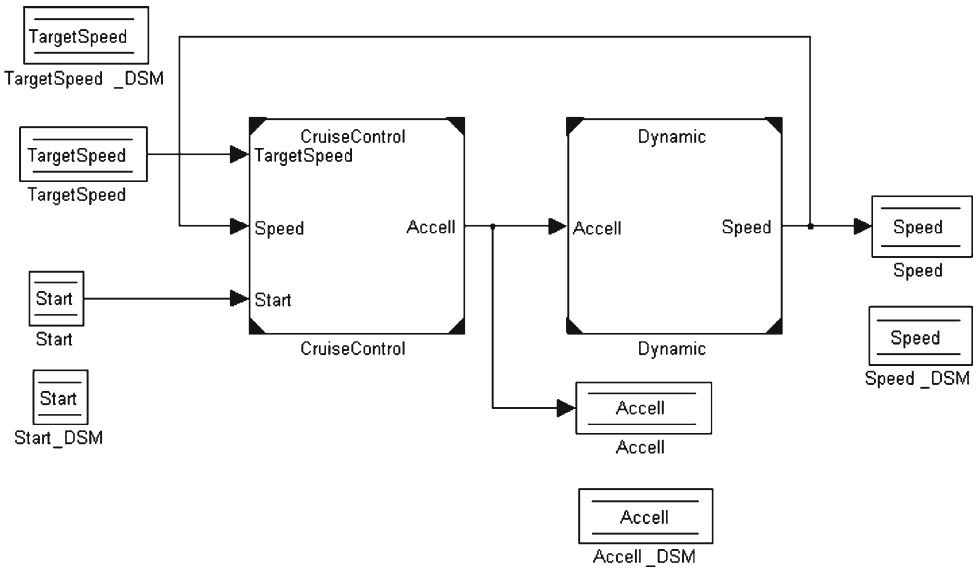

Fig. 9 Simulink block diagram of the CCS algorithm

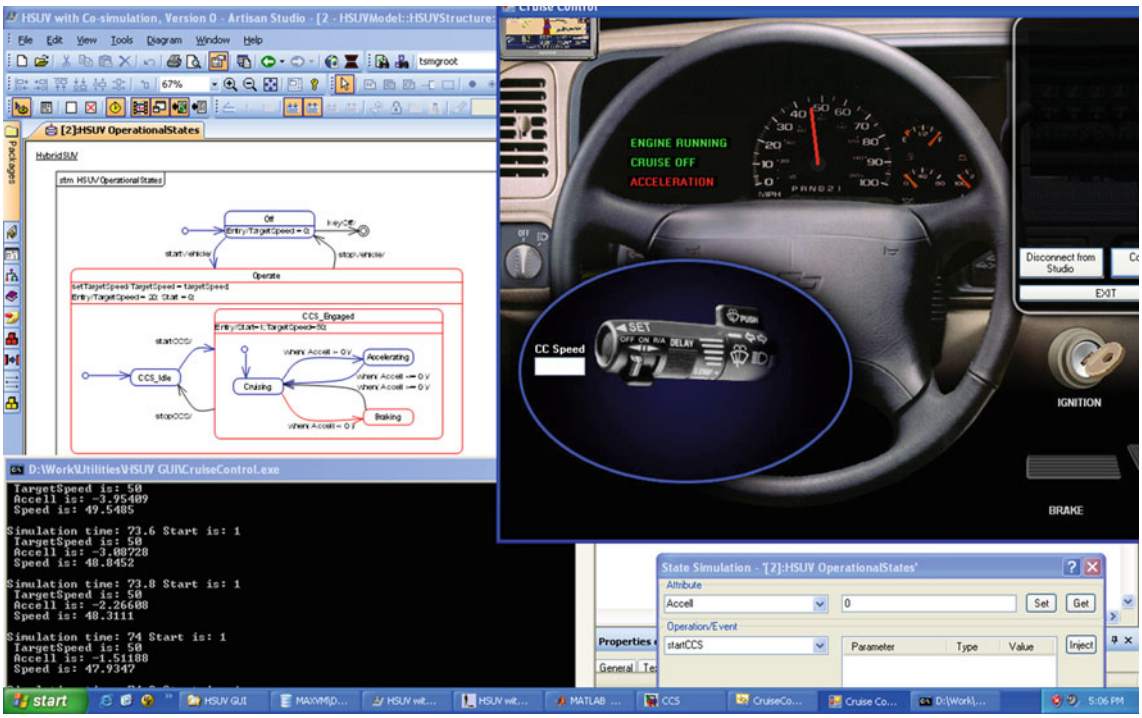

Fig. 10 Simulation snapshot of a Cruise Control System for a HSUV

by the user, but calculated in a real-time way by the algorithm as implemented in Simulink. A simulation snapshot is shown in Fig. 10. The initial speed is 0, then it is set to the value 50 km/h.

7 Conclusion and future directions

This article proposes an approach that combines the Matlab Simulink (for continuous time modeling) with the SysML UML profile (for discrete event modeling) to provide a model-driven continuous/discrete co-simulation framework for complex heterogeneous systems. The proposed co-simulation framework is simple and intuitive since it adopts a code-in-the-loop schema where optimized C/C++ code is generated from Simulink and SysML models automatically by reusing existing tools for code generation for SysML (Artisan Studio) and Simulink. The framework is an Artisan Studio add-in but it is still in an experimental phase.

As future directions, we aim at improving the co-simulation framework for allowing a more interactive debugging

tool and automatic generation of application-specific graphical user interfaces. We have been working also on the evaluation of our approach in large scenarios with the development of (mostly confidential) in-house case studies and best practices in conjunction with Atego partners for demo/training purposes.

As future work, we will also investigate in more detail the relation between the SysML and other UML profiles such as MARTE to find more synergies and integration paths along the system development process and promote their joint use in a unified co-simulation framework. In MARTE, for example, time modeling is a core concern. The basic idea would be therefore to adopt both SysML and MARTE, as MARTE can complement SysML for timing aspects. Some interesting contributions in this direction are the work in [6] where the authors assess possible strategies for combining the SysML and MARTE profiles in a common modeling framework, and the work in [12] where some guidelines are presented on how to use the SysML and MARTE profiles for design space exploration of real-time embedded systems. Also the SATURN EU Project is investigating about SysML and MARTE integration [13]. Another EU Project that deserves further investigation is MODELISAR [1] for the purpose of coupling different simulation tools and for the availability of an open exchange format for simulation models of embedded software in vehicles.

References

1. Modelisar, ITEA2 European project. http://www.modelisar.com/
2. ATESST project. http://www.atesst.org (2007)
3. Bouchhima, F., Briere, M., Nicolescu, G., Abid, M., Aboulhamid, E.M.: A SystemC-Simulink co-simulation framework for continuous-discrete-events simulation. In: Proceedings of IEEE 2006 International Behavioral Modeling and Simulation Workshop, pp. 1–6, San Jose, CA (2006)
4. Brisolara, L.B., Oliveira, M.F., Redin, R., Lamb, L.C., Carro, L., Wagner, F.: Using UML as front-end for heterogeneous software code generation strategies. In: DATE, pp. 504–509. IEEE, New York (2008)
5. ESL Now survey. http://www.esl-now.com (2005)
6. Espinoza, H., Cancila, D., Selic, B., Gérard, S.: Challenges in combining SysML and MARTE for model-based design of embedded systems. In: Paige, R.F., Hartman, A., Rensink, A. (eds.) ECMDA-FA. Lecture Notes in Computer Science, vol. 5562, pp. 98–113. Springer, Berlin (2009)
7. Reichmann, C. et al.: Model level coupling of heterogeneous embedded systems. In: Proceedings of 2nd RTAS Workshop on Model-Driven Embedded Systems (2004)
8. Extessy. Exite tool. http://www.extessy.com/
9. Real-Time Innovation. Constellation framework. http://www.rti.com/
10. OMG, UML Profile for Modeling and Analysis of Real-time and Embedded Systems (MARTE), ptc/08-06-08 (2008)
11. Mueller, W., Rosti, A., Bocchio, S., Riccobene, E., Scandurra, P., Dehaene, W., Vanderperren, Y.: Uml for esl design: basic principles, tools, and applications. InL: ICCAD'06: proceedings of the 2006 IEEE/ACM international conference on computer-aided design, pp. 73–80. ACM, New York, NY, USA (2006)
12. Mura, M., Murillo, L.G., Prevostini, M.: Model-based Design Space Exploration for RTES with SysML and MARTE. In: Forum on specification and Design Languages, FDL, Sept 23–25. IEEE, Stuttgart (2008)
13. SATURN European project: SysML bAsed modeling, architecTUre exploRation, simulation and syNthesis for complex embedded systems. http://www.saturnsysml.eu/
14. Schmidt, D.C.: Guest editor's introduction: model-driven engineering. IEEE Comput. **39**(2), 25–31 (2006)
15. Sjöstedt, C.-J., Shi, J., Törngren, M., Servat, D., Chen, D., Ahlsten, V., Lönn, H.: Mapping Simulink to UML in the design of embedded systems: investigating scenarios and structural and behavioral mapping. In: OMER4 Post-proceedings (2008)
16. OMG, SysML, formal/2007-09-01. http://www.omgsysml.org/ (2007)
17. SystemC Language Reference Manual. IEEE Std 1666 (2006)
18. OMG. The Unified Modeling Language (UML), v2.2. http://www.uml.org (2009)
19. OMG, UML Profile for SoC Specification, v1.0.1 (2006)
20. Vanderperren, Y., Dehaene, W.: From UML/SysML to Matlab/Simulink: current state and future perspectives. In: Gielen Georges, G.E. (ed.) DATE, p. 93. European Design and Automation Association, Leuven, Belgium (2006)