Heterogeneous systems co-simulation: 
a model-driven approach based on 
SysML State Machines and Simulink

Massimo Bombino\(^1\) Matthew Hause\(^2\) Patrizia Scandurra\(^3\)

\(^1\) Atego, Peschiera Borromeo (MI), Italy - massimo.bombino@atego.com
\(^2\) Atego, Cheltenham, United Kingdom - matthew.hause@atego.com
\(^3\) DIIMM - University of Bergamo, Italy - patrizia.scandurra@unibg.it

Abstract. The increasing complexity of real-time systems makes their simulation and validation a demanding task. For most systems, the simulation has to take into account both continuous time and discrete events, and the challenge is to create a co-simulation environment that allows synchronization and interaction between the two worlds. This paper proposes a model-based \textit{code-in-the-loop} co-simulation framework that relies on the OMG SysML (for discrete events and states modeling with State Machine Diagrams) as implemented in the Artisan Studio tool, and on the industry de-facto standard Matlab Simulink tool (for continuous time modeling and simulation). The main feature is the automatic generation of optimized code, allowing real-time simulation that may eventually run natively on a target for embedded systems. Additional features to enhance the simulation effectiveness, like remote graphical animation and control of the state diagrams evolution, are also supported.

1 Introduction

During the design and development of a complex modern system, engineers have to face some difficulties as they have to bring together different aspects from different domains: mechanical, electronic, telecommunications and real-time software aspects. When a new system is designed or updated, often the core is a special algorithm: think about the Flight Control System (FCS) of a flying vehicle (e.g. a jet), or the Radio Transmission Algorithm like W-CDMA for a wireless communication system like UMTS or OFDMA for WiMAX. In this case, the algorithm specialists often start to work before the system engineers, since they have to simulate in advance all the details of the algorithm with a separate tool, usually Matlab Simulink, in order to validate the underlying technology. The Simulink model, possibly composed by sub-modules, is typically a continuous time-description of the algorithm, or the \textit{plant}; but usually it is only a small part of the overall system (think again about the FCS of a jet). On the other hand, system-level events are usually asynchronous and discrete; consider, for example, user interactions, threshold passing, alarms, etc. The system status is usually described by a \textit{state model} using a state-like formalism such as Finite
State Machines or UML/SysML State Machine Diagrams, where the occurrence of discrete events triggers transitions from one state to another.

Due to this multifaceted nature, modern system development requires advanced “collaborative engineering” from different disciplines (such as software engineering, hardware-software co-design, and system modeling, to name a few). Recently, Model-Based Engineering (MBE) 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 higher degree of abstraction, and therefore appear the most promising methods to tackle the complexity of heterogeneous systems. Sooner or later, system engineers start to build a model of the entire system with the OMG standard SysML, that is starting to be widely adopted. The description of the structure of the system with SysML is done with Block Definition Diagrams (BDD) and Internal Block Diagrams (IBD) to model the collaborative and hierarchical structure of a system in terms of modular units called blocks. The behavior may be described in terms of Activity Diagrams, Sequence Diagrams, and State Machine Diagrams (SD). The latter are very important, because they provide a well-consolidated formal description of the states (and sub-states) of a system, and they can be easily animated at code-level since their synthesis into code is easily performed by exploiting well-known generation patterns (such as the RunToCompletion, state pattern, stable table pattern, etc.). For requirement traceability, SysML Requirement Diagrams allow the system engineer to model requirements and relate them to other model elements that satisfy or verify them. Finally, the constraints of the system may be described by Parametric Diagrams (ParD) that allows us to specify system parameters and relate them to each other.

More synergies and integration are, however, necessary before collaborative model-based IDEs become available for use in multi-disciplinary contexts. In particular, research in the continuous/discrete co-design and co-simulation area is still undergoing. The idea of co-simulation is to bring together the Continuous Time (CT) together with Discrete Events (DE), in order 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.

![Co-Simulation Framework](image-url)
This paper proposes a code-in-the-loop co-simulation framework⁴ for continuous/discrete systems based on two well-known modeling notations: SysML (for the discrete parts) and Simulink (for the continuous parts). The framework relies on MBE principles allowing automatic C/C++ code generation from 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 values between the CT/DE simulators and cover time synchronization aspects. The resulting code is automatically compiled for a specific target (a host PC or an embedded target) and run at real-time or scaled real-time.

2 Co-Simulation Framework

Figure 1 shows the overall architecture of the proposed co-simulation framework. On the left side, a Simulink model is used to describe the continuous time aspects of the system, or the main algorithm. The simulation may be self-contained, but usually it has some inputs and outputs from the remaining parts of the system. Data are provided in form of vectors or stored signals, and the output is recorded in form of vectors or graphs. On the right side, Artisan Studio is used to develop the SysML State Machine Diagrams (SDs) describing a state model of the system⁵.

The automatic code generation of MBE is one of the most promising techniques used deeply in our approach. The code generated by the Simulink Real-Time Workshop (RTW) tool usually is used “as it is”, but could be easily interfaced to another simulation environment in order to control the simulation and provide input and outputs. The Artisan Studio ACS is a new real-time synchronizer that keeps model and code synchronized according to a choice of Generator DLLs that range several different languages and applications. The default code generation for SysML State Machine Diagrams is based on the RunToCompletion semantics, in the standard C++ Generator DLL. One of the best aspect of the ACS is that is completely customizable, through the Artisan Studio Transformation Development Kit (TDK), allowing the creation of code generation patterns and model-to-model transformations. The languages natively supported in TDK are SDL (Syntax Definition Language) and RSN (Reverse Syntax Notation).

⁴ In a code-in-the-loop schema, co-simulation is based on a target implementation language that is used as common execution language for both models.

⁵ A state model is a standard way to describe the behavior of a discrete system. SysML SDs are formal enough to be used for this purpose and, in contrast to the Simulink/Stateflow formalism that is also suitable for the same goal, are well integrated in the overall SysML-based system design activity with other SysML diagrams (BBDs,IBDs, ParDs, etc.). Other SysML behavioral diagrams, such as Activity Diagrams, are not yet formal enough for simulation purposes, except Sequence Diagrams that may be used to provide input or output vectors to SDs simulation.
In this way, the default code generation mechanism for SysML State Diagrams may be changed in order 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.), with a split in Platform Independent Model (PIM) and Platform Specific Model (PSM) according to the OMG Model Driven Architecture (MDA) vision [5]. In particular, we exploited the code generation feature of TDK for the generation of “glue code” for most of the aspects of the co-simulation framework. Starting from a standard C++ Generator DLL with animation capabilities, all the functionalities are built through model transformations written in SDL.

![Fig. 2. The State Machine Diagram for HSUV and CCS](image)

The final code can be generated for a normal Windows PC or for a specific embedded target (currently supported OSs are Linux and VxWorks), and compiled for the specific architecture. The compilation and execution facilities for the simulation are automatically provided, so that the system engineers do not have to necessarily deal with the code. An appropriate context-menu is available when editing SysML SDs with the commands for compilation, verification and running of the simulation. Additional features are also supported such as graphical animation of the SysML SDs on the host PC, connected through a TCP/IP link to the target performing the simulation, and the opportunity to send or receive events to/from the simulation environment from a graphical interface.

As complete case study, a cruise control system for a Hybrid Sport Utility Vehicle (HSUV) has been developed. The Simulink implementation of the core Cruise Control System (CCS) algorithm is co-simulated with the HSUV SysML SD (see Fig. 2) that represents the basic operational states of the vehicle. Once the vehicle started and the CCS is engaged, the state of the machine (ranging in the set \{**Cruising**, **Accelerating**, **Braking**\}) is no more static or forced by...
the user, but calculated in a real-time way by the algorithm as implemented in Simulink. A simulation snapshot is shown in Fig. 3.

![Simulation snapshot of a Cruise Control System for an HSUV](image)

**Fig. 3.** Simulation snapshot of a Cruise Control System for an HSUV

### 3 Related work

Experiments to integrate the Mathworks 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 [8], two different approaches for coupling UML and Simulink have been proposed so far: (i) **model-in-the-loop** co-simulation via an intermediate coupling bus that allows the two simulations (Simulink and UML simulations) to communicate and (ii) **code-in-the-loop** co-simulation based on a target implementation language that is used as common execution language for both models. As an example of the first co-simulation approach, the Exite tool [3] allows the coupling of a Simulink model with Artisan Studio. The second approach is adopted for example in the Constellation framework [4] and in the General Store integration platform [2]. Both approaches vary in the provided abstraction and effective integration. Both solutions focus on the use of different modeling languages to specify each system module. The first co-simulation approach requires special attention to the synchronization aspect. On the other hand, specific development frameworks which ease the creation of executable code (usually C++) from UML and Matlab/Simulink allow faster simulation speed.

The work we proposed here is an example of the second co-simulation approach, as the integration is effectively made at code-level. Comparing with the presented related work, our framework: (1) proposes 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-based approach that relies on an small set of specific stereotypes (an extension of the SysML language) and transformation rules applied to the input SysML model; (3) supports the native running on target for embedded systems (no instances of Simulink and Studio are required); (4) provides remote model debugging/animation features.

The work in [1] is the most similar to our one. Their co-simulation approach relies on Simulink for the continuous simulation and on SystemC\(^6\) for the discrete simulation. However, they do not rely on a MBE approach; indeed, SystemC is a code-based formalism too specific to the System-on-Chip (SoC) domain and as modeling language may not cover all aspects required in a multidisciplinary domain to describe complex heterogeneous systems as the SysML may do.

The Ptolemy project\(^7\) studies modeling, simulation, and design 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 do not rely on standard notations, while our framework is completely SysML-based (OMG standard) and Simulink-based (de-facto standard).

In [6], 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\(^8\) in the automotive domain to study the feasibility of defining an integration with our approach (that relies, instead, on the SysML SDs).

References


\(^6\) SystemC \cite{7} 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.

\(^7\) http://ptolemy.eecs.berkeley.edu/

\(^8\) www.atesst.org