COUPLING SIMULINK AND UML MODELS

Jozef Hooman [1], Nataliya Mulyar [2], Ladislau Posta [2]
[1] *Embedded Systems Institute & University of Nijmegen*
Address: Laplace-building 0.10, P.O. Box 513, 5600MB Eindhoven, the Netherlands
Phone: +31 (0) 40 247 8222, e-mail: jozef.hooman@embeddedsystems.nl
[2] *Eindhoven University of Technology, the Netherlands*

Abstract: The general aim of this work is to support the multi-disciplinary development of real-time embedded systems by combining tools of different disciplines. As a concrete example, we have coupled a UML-based CASE tool (Rose RealTime) and Simulink to allow simultaneous simulation. Main conceptual problem is to establish a common notion of time. We have implemented a first prototype in which this has been solved by using the simulation time of Matlab/Simulink also for Rose RealTime and by extending the UML model with assumptions about execution times.

Keywords: multi-disciplinary modeling, real-time, simulation, discrete-time models, continuous-time models

1 INTRODUCTION

The development of embedded real-time applications, including transportation, typically involves several disciplines, such as electrical engineering, mechanical engineering, and software engineering. Although these disciplines are tightly coupled in the considered embedded systems, their development is often a rather sequential, mono-disciplinary, process. Typically, first the mechanical part is designed, next the hardware infrastructure is fixed, and finally the embedded software is developed. This approach can create large problems, especially for the software engineers. For instance, choices about the placements of sensors (and implicitly the occurrence of interrupts), control rates, control delays, hardware, etc., have a strong influence on the complexity of the software. Moreover, usually many implicit assumptions are made, which first become visible at system integration. This easily leads to non-optimal solutions.

Within each discipline, a common solution is the frequent use of models to detect problems as early as possible. For instance, in the software domain a lot of effort is put on model driven development, based on UML models (Booch *et. al.* 1999). Moreover, mono-disciplinary modeling is usually supported by tools that allow some form of execution or simulation. Lacking, however, is the possibility to combine tools of different disciplines and to investigate the mutual influence of modeling choices. Our aim is to couple currently used tools to allow simultaneous simulation of models from different disciplines.

The work described here is mainly part of the Boderc project*, in collaboration with the company Océ, a producer of high-volume printers and copiers, but the proposed solutions could also be applicable in other domains such as avionics and automotive.

Given the collaboration with Océ, we have implemented a coupling between the UML-based CASE tool Rose RealTime (currently renamed to Rose Technical Developer) of IBM Rational and Matlab/Simulink of The Mathworks. Rose RealTime (Rose-RT) supports the ROOM methodology (Selic et. al., 1994) for the development of software for real-time reactive systems. It is used at Océ to define a reusable software architecture, which is instantiated for a particular printing/copying machine. Next the tool allows the

* This work has been carried out as part of the Boderc project under the responsibility of the Embedded Systems Institute. This project is partially supported by the Netherlands Ministry of Economic Affairs under the Senter TS program. The first author is partially supported by IST-2002-33522 project OMEGA.

generation of code for a particular target platform, thus obtaining a direct connection between a model and the generated code. Simulink is used at Océ to model the mechanical layout of the machine and to experiment with, for instance, the shape and the length of the paper path, the placement of motors and sensors, and the paper speed.

As an example, our coupling allows the combination of a continuous-time model of a physical dynamical system in Simulink with a discrete-time control algorithm in Rose-RT, as depicted in *Fig. 1*.

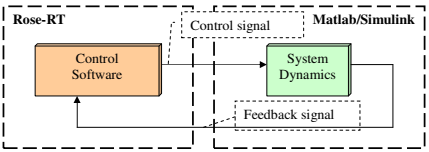

Fig. 1 Combined Models

By establishing a proper notion of simultaneous simulation of these models, one can quickly investigate the effect of changes in the control strategy, the software execution times, or the effect of different system characteristics.

Realizing the desired tool coupling is far from trivial. The two main challenges are:
(1) *Conceptual correctness*. The coupling should be such that the simultaneous simulation of models in both tools gives meaningful results. In particular, this means that there must be a common notion of simulation time in combination with a proper exchange of data and messages. For instance, control data computed by the UML model is to be used in the Simulink model at the right moment in time, taking the duration of the computation into account. Moreover, time-outs generated by timers in the Rose-RT model should correspond to the simulated time in Simulink.
(2) *Technical implementation*. The coupling software should be properly designed to allow, for instance, a change to another UML tool without too much effort. Moreover, suitable ways should be found to allow the tools to communicate and to run a simulation mode simultaneously. The current version of the coupling works in a Windows environment with Matlab release 13, Simulink version 5.0, and Rose-RT version 6.5.341.0.

Such a coupling between Rose-RT (or similar real-time UML tools) and Simulink models has not been realized before. Related is the work on the High Level Architecture (HLA) (Dahmann *et. al.* 1997), a general-purpose architecture for the coupling of simulation tools. However, HLA cannot be used for our purpose, because UML-based CASE tools, such as Rose-RT do not fit into the HLA framework since they do not have the required simulation mode with a well-defined notion of simulation time.

An alternative solution to the modeling of mixed discrete-continuous systems has been followed in the HyROOM approach (Stauner *et. al.* 2001), where the ROOM/Rose-RT notation has been extended with continuous elements. The main parts of the resulting tool have been mapped into HyCharts (Grosu *et. al.* 1998), a formal framework for hybrid systems. Along this line, there are several formal approaches that allow checking properties of hybrid systems (modeling both discrete and continuous aspects), such as HyTech (Henzinger *et. al.* 1997) and Checkmate (Clarke *et. al.* 2003). Another approach is the use of generic modeling environments, such as Ptolemy and the Generic Modeling Environment (GME), that allow the development of domain specific modeling tools. Our aim, however, is to allow engineers to continue working with their well-known mono-disciplinary tools, without having to redo their modeling work in some multi-disciplinary modeling environment.

The rest of this paper is structured as follows. Sections 2 and 3 contain a very brief presentation of the tools used (Rose-RT and Simulink, respectively); describing them only as far as needed to understand the coupling. The main concepts of the coupling are explained in Section 4. Details of the implementation are given in Section 5. Concluding remarks can be found in Section 6.

2 ROSE REALTIME

Rose-RT is a UML-based CASE tool for the development of complex reactive software. Typically, a UML model in Rose-RT consists of a number of active objects, also called *capsules*, which communicate by sending and receiving messages via *ports*. Messages may have different priorities. A port must refer to a *protocol*, which represents a set of messages that can be exchanged between capsules. The behavior of a

capsule is modeled by means of a hierarchical state diagram. Transitions in a state diagram are triggered by the receipt of messages or time-outs. Actions on a transition may change local variables, send messages, or set timers.

Given a complete model, the Rose-tool can generate code for a specific target platform, using the characteristics of that platform. Model execution is based on a so-called Service Layer which provides general services, e.g. it contains controllers, which are responsible for queuing and delivering messages among capsules, and timing services that can be used in the generated code.

The implementation of the Service Layer depends on the target platform on which the program should run. Hence, at the code generation and compilation step for a Rose-RT model, the toolset links the user-defined code with a services library for the particular platform on which the model is intended to run.

The Service Library also contains services for concurrency control and thread management. Capsules can belong to different logical threads. Logical threads are mapped to a set of concurrent physical threads defined by the developer of the Rose-RT model. No other capsules in a thread can execute until the currently executing capsule returns control to the main loop of that thread. However, other capsules on other physical threads may be executing concurrently.

Each thread has a separate message queue and its own controller object that is responsible for queuing and delivering messages among capsules in this thread. This controller object contains the basic message delivery and processing loop. The underlying operating system is responsible for switching control among active physical threads. The operating system may pre-empt one physical thread in the middle of execution to switch to another physical thread. Each thread can be assigned a separate priority, so that the designer has some control over the scheduling.

Message processing, which is provided by the Services Library, is based on the following steps. During start-up, the initialization message is the message with the highest priority. When a capsule processes the initialization message, the capsule's initial transition segment is executed. During the main processing loop the controller object takes the next highest priority message from the message queues and delivers it to the receiver capsule and invokes that capsule's behavior to process the message. Each capsule processes the current message to the completion of the transition chain. This is referred to as the *run-to-completion* semantics. When the capsule has completed processing a message, it returns control to the controller. The controller continues this loop until there are no more messages to be processed.

One *step* in Rose-RT is associated with processing the next message of the highest available priority. A step terminates when all actions associated with the respective message are performed.

There are two ways of testing the generated and compiled code. The first way is to run the executable on the intended platform. The second way is to execute the model step-by-step on a simulated platform, but then correct timing is not guaranteed and only the reactive response to messages can be tested.

The Timing Service of the Rose-RT Services Library provides the model developers with general-purpose timing facilities based on both absolute and relative time. The precision of the Timing Service depends on the granularity of the timing supported by the underlying operating system. The Timing Service does not guarantee absolute accuracy, since it is platform dependent. This means that intervals between timer creation and timer expiration can take slightly longer than specified, and events scheduled for a particular time may in fact happen slightly after the actual time has occurred.

3 MATLAB/SIMULINK

This section contains a short description of Simulink of The MathWorks, based on the tool documentation. A Simulink model is represented graphically by means of a number of interconnected *blocks*. Lines between blocks connect block outputs to block inputs. Blocks may have states, which may consist of a discrete-time and a continuous-time part.

The output of a block is computed by an *output function*, based on its input and its current state and time. Similarly, an *update function* calculates the next discrete state. A *derivative function*

relates the derivatives of the continuous part of the state to time and the current values of the inputs and the state.

Blocks can be built from a large number of predefined library blocks, or they can be implemented by an S-function, which can be written in MATLAB, C, C++, Ada, or Fortran.

During the simulation of a Simulink model, the outputs, inputs and states are computed at certain intervals, from a start time to an end time, as specified by the user. The successive states of a system are computed by a so-called *solver*, a Simulink-specific program. Since no solver is suitable for all models, there are several types of solvers. The solvers use numerical integration to compute the continuous states of a system from the state derivatives specified by the model. Each solver uses a different integration method, allowing the selection of the most suitable method for a particular model.

The successive time points at which the states and outputs are computed are called *time steps*. The length of time between steps is called *step size*. The step size depends on the type of the solver used, the characteristics of the Simulink model, and the existence of discontinuities of the continuous states (Simulink checks for such discontinuities – this is called zero crossing detection – and if it detects one within the current step, the precise time at which zero crossing occurs is determined and additional time steps are taken).

There are several types of solvers. *Fixed-step solvers* use a fixed step size. *Variable-step solvers* change the step size during simulation. They reduce the step size to increase accuracy when states are changing rapidly and increasing the step size to avoid taking unnecessary steps when states are changing slowly. This requires some additional computation each step, to determine the step size, but can reduce the total number of steps and hence the duration of the simulation. For purely discrete models there are *discrete solvers*. *Continuous solvers* compute continuous states using numerical integration. Simulink provides an extensive set of fixed-step and variable-step continuous solvers, each implementing a specific numerical integration technique for solving the ordinary differential equations that represent the continuous states of dynamic systems.

The solvers monitor the error at each time step; they compute the local error, which is the estimated error of the computed state values. If the local error is greater than the acceptable error for any state, the solver reduces the step size and tries again.

Simulation of a Simulink model starts with the initialization phase, where e.g. library blocks are incorporated, block parameters are evaluated, memory is allocated and the execution order of the blocks is determined. Next, Simulink enters a simulation loop, consisting of *simulation steps*. During each simulation step, Simulink executes all blocks of the model in the order determined during initialization. This execution order does not change during the simulation. For each block, Simulink calls functions that compute the block's states, derivatives, and outputs for the current sample time. This continues until the simulation is complete.

4 MAIN CONCEPTS OF THE COUPLING

In this section, the main decisions taken to establish a correct coupling are presented, namely, the notion of time in Section 4.1 and the global coupling architecture in Section 4.2.

4.1 Notion of time

The most important decision concerns the notion of time to be used for the simulation. Observe that the timing of Rose-RT is strongly coupled to the timing service of the operating system of the target system on which the model is running. Moreover, timing is not respected in the step-by-step simulation. Hence, we concluded that the timing of Rose-RT is not suitable for our purpose and decided to use the notion of simulated time of Simulink instead. The alternative is to use a separate, independent, notion of time, but this would also require new implementations of solvers, redoing a lot of things already available in Simulink.

To be able to establish a proper notion of simulation time, which faithfully reflects the execution of both models, somehow the execution time of the transitions in the Rose-RT model has to be taken into account. We assume that this information is available, representing an assumption on the underlying platform.

4.2 Global coupling architecture

Another decision to be taken is the global architecture of the coupling. Instead of a tight coupling, we decided to use a more loosely coupled architecture by introducing a third component called *Multidisciplinary Coupling Tool* (*MCT*), as shown in *Fig. 2*. Observe that each tool contains an add-in, which is responsible for the communication with the MCT component.

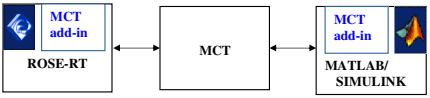

Fig. 2 Loosely Coupled Architecture

By introducing such an MCT interface, the modeling tools do not need to know about each other and it becomes much easier to change. For instance, to switch to another UML-based CASE tool. Moreover, it makes it easier for the engineers to establish a coupling without knowing much about the details of the models of the other discipline.

To obtain proper timing of the UML models, we have redefined the timing service of Rose-RT such that it gets the current notion of time from the MCT component, which passes on the notion of time it receives from Simulink.

5 DETAILS OF THE COUPLING

This section contains more implementation details of the realized coupling. *Fig. 3* shows a rather detailed module architectural view of the implementation, showing for instance in more detail how the timing of a UML model is obtained from the MCT. The *Original Rose-RT model* and the *Original Simulink model* depict the original models supplied for coupling.

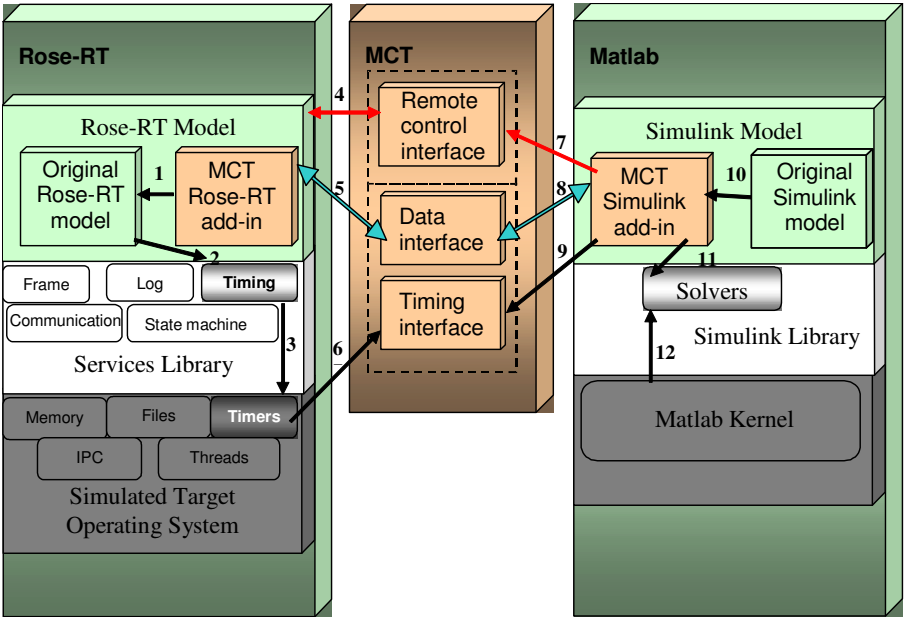

Fig. 3 Module Architecture View

Below we describe the main parts, the Rose-RT layers in Section 5.1, the MCT in Section 5.2, and the Matlab/Simulink layers in Section 5.3, referring to arrows in *Fig. 3*.

5.1 Rose-RT layers

The Rose-RT component consists of three layers: the Rose-RT Model layer, the Services Library layer, and the Simulated Target Operating System layer.

The *Rose-RT Model* layer consists of the *Original Rose-RT model* and the *MCT Rose-RT add-in,* which is responsible for the external communication (arrow 1 in *Fig. 3*). In general, a Rose-RT model may communicate with external applications through external ports only. An external port can only identify the presence of a Rose-RT-specific type signal. This actually means that data cannot be send to an external port of a capsule. However, we can associate an external port with a signal, which notifies that the data supplied by the Simulink model is available in some data storage (here located in the *MCT* component). Consequently, every different data unit should be associated with a separate external port. The *MCT Rose-RT add-in* contains the collection of all these external ports.

A Rose-RT model has access to classes of the *Services Library* layer. For example, whenever a model needs to use the timing service, as in the case of a timer creation, it uses the methods of classes that implement the *Timing Service* of the *Services Library* (see arrow 2). Arrow 3 shows that the *Timing Service* of the *Services Library* queries the *Timers* of the *Simulated Target Operating System*. These *Timers* (of the *Simulated Target Operating System*) use the *Timing interface* of the MCT (see arrow 6) in order to access the simulation time of Simulink instead of the one originally used by Rose-RT.

5.2 MCT

The MCT component consists of three interfaces: a *Remote control interface,* a *Data interface*, and a *Timing interface*.
The *Remote control interface* allows starting, stopping and controlling the execution of the Rose-RT model in step-by-step mode (arrow 4). This functionality can be accessed by the *MCT Simulink add-in* (arrow 7).
The *Data interface* serves as storage for the data that has to be exchanged between the Rose-RT and Simulink models, including the timing delays associated with the execution of transitions in Rose-RT (arrows 5 and 8). For example, data calculated by Simulink is set in the *MCT* to be available for Rose-RT. After the *MCT* notifies Rose-RT about the data availability through the *Remote control interface,* Rose-RT can access the data in the *MCT* storage. The other direction of data transfer proceeds similarly. The *Data interface* also plays an important role in the time synchronization process. After executing a transition, the assumed execution time is sent to the *MCT Simulink add-in* using the *Data interface.*
The *Timing interface* keeps track of the simulation time. It represents an intermediate clock, which is updated with the value of the Simulink simulation time (arrow 9) and which is regularly sampled (before a step in Rose-RT is executed) by the *Timers* of the *Simulated Target Operating System* (arrow 6).

5.3 Matlab/Simulink layers

The Matlab/Simulink component has three layers: the Simulink Model layer, the Simulink Library layer, and the Matlab layer.

The *Simulink Model* layer contains the *Original Simulink model*, extended by the *MCT Simulink add-in* (see arrow 10). The *MCT Simulink add-in* is actually the driver of the simulation, and therefore the *Original Simulink model* depends on it.

Timing of Simulink does not depend on the platform on which the tool runs, but is defined by the *Solvers*. The *MCT Simulink add-in* takes the value of the current simulation time provided by one of the *Solvers* (see arrow 11) and passes it to the *Timing interface* (arrow 9).

The *MCT Simulink add-in* uses the functions of the *Remote control interface* to send events to the *MCT Rose-RT add-in*, and to drive the Rose-RT execution in the step-by-step mode (arrow 7). The command to perform a step in Rose-RT should always be preceded by an update of the Rose-RT time in order to keep the clocks of Rose-RT and Simulink synchronized. After each step performed by Rose-RT, the *MCT Simulink add-in* gets the new data, including the assumed time duration of the executed transition(s). This data is obtained through the *Data interface*. To ensure that Simulink takes this execution delay into account, we have designed a block diagram in which the original Simulink model should be inserted, as depicted in Fig. 4.

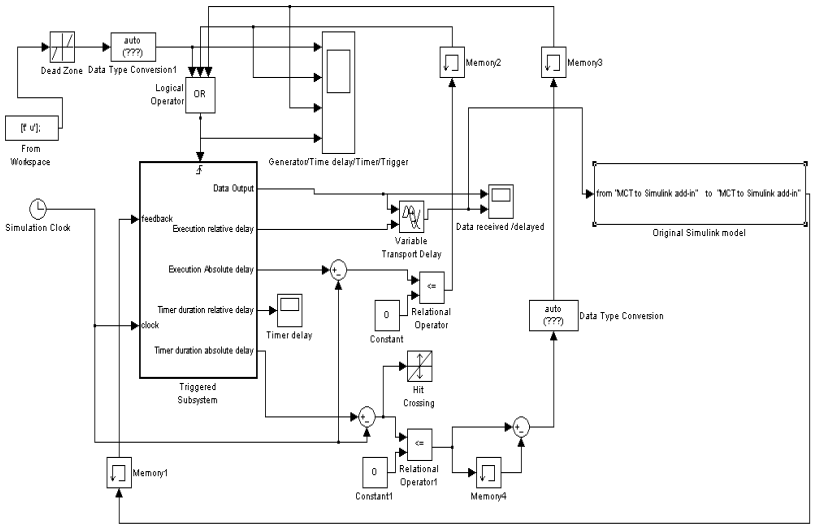

Fig. 4 Extended Simulink model

This extended Simulink model contains a block called *Triggered Subsystem* which is enabled each time a trigger is detected. There are three types of triggers possible:

(1) Initial trigger (when the simulation starts); represented by the blocks *From Workspace, Dead Zone*, and *Data Type Conversion1.*

(2) Execution delay trigger; generated when the Simulink simulation time is advanced with the value of the execution delay of a transition executed in Rose-RT. It is represented by the blocks *Clock, Relational Operator*, and *Memory2*.

(3) Timer expiration trigger. This trigger is generated when the Simulink simulation time is advanced with a value equal to the duration of the timer, as requested by a timer setting in Rose-RT. This trigger is represented by the blocks *Clock, Relational Operator, Hit Crossing, Memory3*, and *Memory4*.

Block *Triggered Subsystem* is shown in *Fig. 5*. Inside this *Triggered Subsystem* one can notice a *MCTblock* which is a user-created block defined by the S-function *MCTSfunction.* This *MCTSfunction* has three outputs, which are passed to the five outputs of the *Triggered Subsystem* block. Output *Execution relative delay* is used for the simulation of delays by the *Variable Transport Delay* block (see *Fig. 4*). This block ensures that data received from the *Data Output* of the *Triggered Subsystem* is provided only after the time specified by the *Execution relative delay.* Output *Execution relative delay* is also used for the calculation of the *Execution Absolute delay*, which is used to generate *the Execution delay trigger* to the *Triggered Subsystem*.

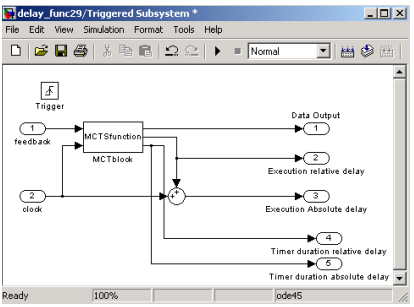

Fig. 5 Triggered Subsystem

Next we describe the behaviour of the S-function *MCTSfunction* defining the *MCTblock*. A standard S-function contains a number of functions that are called by Simulink. For instance, during a simulation step, Simulink calls *mdlUpdate()* to update discrete states, *mdlDerivatives()* to calculate derivatives, and *mdlOutputs()*

to calculate the outputs of a block. In our case, the *mdlOutputs()* function defines when and how the synchronization of the Rose-RT and Matlab/Simulink models is performed. The order in which commands are given within the *mdlOutputs()* function is crucial for the execution and should be strictly followed:

1. Set the clock in the *Timing interface* to the current simulation time. This is needed for synchronizing the Rose-RT clock with the Simulink clock.
2. Give a command to Rose-RT to perform one step. By doing this, Rose-RT will become active and responsive to external events.
3. Send an external event, if no timer was started in Rose-RT, which will trigger a transition in the Rose-RT model.
4. Read the data, execution delay and timer duration from the Data interface and pass it to the output of the *MCTblock.* If these were not set during the execution of the step in Rose-RT the previous values will be read.
5. Take the input of the *MCTblock,* provided by the *Original Simulink model* and put it in the *Data interface* to make it available for Rose-RT.

Finally, we propose the selection of a continuous variable-step solver for the simulation, to ensure an effective way of calculating data and determining critical points in the simulation.

6 CONCLUDING REMARKS

The current version of the coupling tool that connects Simulink and Rose-RT is a first prototype that can be used to investigate the main principles and to experiment with examples. It is has been tested an a few small examples, but more experiments are needed to investigate the behavior for various types of solvers and models and to get more confidence in the correctness of the simulations. Moreover, we have to apply the coupling to large existing models from industry to investigate the feasibility and usefulness of such a simultaneous simulation and to investigate the performance for complex systems. Future work also includes the removal of a few simplifications that have been made to obtain a first prototype quickly. For instance, at the moment only one timer is allowed in the UML model and a preliminary version of a timer queue has not yet been tested.

Acknowledgements We would like to thank the members of the Boderc project for constructive discussions, useful hints for the realization of the coupling, support on the design of Matlab/Simulink models, and constructive comments on draft versions of the current paper.

LITERATURE

Booch, G., J. Rumbaugh and I. Jacobson. (1999). *The Unified Modeling Language User Guide*, Addison-Wesley.

Clarke, E.M., A Fehnker, Zhi Han, B. Krogh, J. Ouaknine, O. Stursberg and M. Theobald. (2003). *Abstraction and Counterexample-guided Refinement of Hybrid Systems.* International Journal of Foundations of Computer Science, Vol 14, Number 3.

Dahmann, J., R. Fujimoto, and R. Weatherly. (1997). *The Department of Defense High Level Architecture* In Proc. of the 1997 Winter Simulation Conference, pp.142-149.

The Generic Modeling Environment, http://www.isis.vanderbilt.edu/Projects/gme/

Grosu, R., T. Stauner and M. Broy. (1998). *A Modular Visual Model for Hybrid Systems. In Proc. of the FTRTFT'98.*

Henzinger, T.A., P.-H. Ho and H. Wong-Toi. (1997). *HyTech: A Model Checker for Hybrid Systems.* Software Tools for Technology Transfer 1:110-122.

The Ptolemy Project, ptolemy.eecs.berkeley.edu/

Rose Technical Developer, www.ibm.com

Selic, B., G., Gullekson and P. Ward. (1994). *Real-Time Object-Oriented Modeling*. John Wiley & Sons.

Simulink of The Mathworks, www.mathworks.com/products/simulink/

Stauner, T., A. Pretschner and I. Péter. (2001). *Approaching a Discrete-Continuous UML: Tool Support and Formalization.* Proc. UML'2001 workshop on Practical UML-Based Rigorous Development Methods -- Countering or Integrating the eXtremists, pp. 242-257.