Correct-by-Construction Transformations across Design Environments for Model-Based Embedded Software Development

M. Baleani*, A. Ferrari*, L. Mangeruca*, A.L. Sangiovanni-Vincentelli*,†,
U. Freund‡, E. Schlenker‡, H.-J. Wolff‡

* PARADES E.E.I.G., via San Pantaleo, 66, 00186 Roma, Italy
† EECS Dept., University of California at Berkeley, CA 94720, USA
‡ ETAS GmbH, Borsigstraße, 14, 70469 Stuttgart, Germany

Abstract

Embedded software design for real time reactive system has become the bottleneck in the market introduction of complex products such as automobiles, airplanes, and industrial control plants. In particular, functional correctness and reactive performance are increasingly difficult to verify. The advent of model-based design methodologies has alleviated some of the verification-related problems by making the code-generation process flow automatically from the model description. Given the relative infancy of this approach, several companies rely upon design flows based on different tools connected together by file transfer. This way of integrating tools defeats the very purpose of the methodology introducing a high potential of errors in the transformation from one format to another and preventing formal analysis of the properties of the design. In this paper, we propose to adopt a formal transformation across different tools and we give an example of this approach by linking two tools that are widely used in the automotive domain: Simulink and ASCET. We believe that this approach can be applied to any embedded software design flow to leverage the power of all the tools in the flow.

1 Introduction

Embedded software design for real time reactive system has become the bottleneck in the market introduction of complex products such as automobiles, airplanes, and industrial control plants. Failures are unacceptable for safety-critical systems, such as transportation systems, and over-engineering is unacceptable for systems with tight cost and resource constraints. Embedded software must interact with physical processes guaranteeing support for hard real-time operation and concurrency. Due to the increasing complexity and shrinking time-to-market, an effective methodology must at all stages involve automatic and semi-automatic support by software tools and favor code re-use.

Model based design is emerging as a solution to embedded software design issues as witnessed by both academic [6] and industrial efforts [5]. The basic tenet of this methodology is to move away from manual coding from informal specifications by capturing embedded software functional and non-functional requirements from abstract mathematical models. Clearly, a mathematical model offers a common ground for a systematic and coherent integration of diverse efforts in system specification, design, synthesis (code generation), analysis (validation), execution (run-time support), and maintenance (design evolution). Today, no single model exists that is agreed upon for embedded software design, and flows are based on different tools and models (e.g., in the automotive domain, UML-based, Simulink, and ASCET models are commonly used to represent designs at different abstraction levels).

Integration is certainly possible at the code level, i.e. using C as a common language. However, code-level integration does have severe limitations. In the model-based methodology, a design is usually represented as a set of components interacting with each other and with the environment where a precise model of computation (MoC) defines the behavior and interaction mechanisms of these modules. The MoC describes how each component performs internal computation, how components transfer information between them and how they relate in terms of concurrency. When integrating executable models, there is much more to tool inter-operability than just the definition

This work has been partially supported by EC IST-2001-34820 (ARTIST), EC IST-2-004527 (ARTIST-II) and NFS ITR CCR-0225610 (CHESS). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

1530-1591/05 $20.00 © 2005 IEEE

of a common syntax or data exchange format: we cannot leave aside how the different MoC relates. While C code can be used effectively to represent the functionality of each component, it can hardly be used to express formally their composition (i.e. communication and concurrency). As a consequence, code-level integration require manual integration to some extent and the need of extensively verifying the correctness of models composition. Last but not least, code-level integration actually prevents the use of analysis and/or optimization techniques that may be applicable to the original source model.

This calls for the integration of tools and design representation using rigorous semantics. An example ofthis approach is [1], an excellent paper about the model-based transformation of a Simulink representation into a Lustre representation. In this paper we present a similar correct-by-construction approach to transformations across design environments. To demonstrate this approach we chose two tools (ASCET and Simulink) that are widely used in the automotive domain. Our choice is motivated by the complexity of the embedded software design problem in today cars where premium cars feature up to 5 Giga-Byte of software. The approach is based on the use of a common formal model, namely the synchronous reactive model of computation, which is used as the common ground to interpret system specifications given with different underlying models. In particular, we build an SR abstraction of any given system specifications. This abstraction removes some specific aspects of the original models and this is the price to pay to build a rigorous transformation. We made sure that the behavior of each component is preserved and retain that degree of coordination (i.e. communication and concurrency) needed to ensure the functionality of the system.

The paper is organized as follows. In Section 2 we briefly review the basic features of the MATLAB/Simulink and ASCET design environments. In Section 3 the synchronous reactive model of computation is introduced as the basis for model transformation between the two environments and in Section 4 the SR interpretation of ASCET and MATLAB/Simulink specification is presented. In Section 5 we demonstrate how ASCET models can be ported to Simulink models and *vice versa* in a consistent way using the common SR semantics. Section 6 presents some final considerations.

2 ASCET and Simulink Design Environments

ASCET [4] supports the *Task Programming Model* (TPM) paradigm for single processor implementations of embedded software controllers. According to TPM, system specifications are given as a collection of software components called *tasks* and an appropriate scheduling policy. The system specification in ASCET is called a *project* and consists of a set of *modules*. Modules instantiate *classes* and define *processes*. Classes are reusable components consisting of variables and methods. Component state is held by variables. Component outputs can be computed by calling methods. Processes are methods defined in modules and can call the methods defined by the classes instantiated within the module. Processes communicate via *messages*, which are shared variables protected via local copy: each process works on a local copy of its input/output messages. The protection mechanism is guaranteed by the underlying operating system (RTOS), which is assumed to be OSEK compliant. Processes are scheduled within tasks that are scheduled by the RTOS on a time or event basis. The scheduling mechanism is based on static priority with preemption.

MATLAB/Simulink [9] is a system specification, verification, simulation and synthesis environment for real-time systems. Its model of computation is discrete-time [2]. As often is the case, the results from the "execution" of the model depend on the scheduling algorithm of the simulator that executes the model thus making the semantics of the model not self-contained. This has three disadvantages: the definition of the Simulink's MoC is $1)$ complex and difficult to comprehend as a whole, $2)$ can be significantly dependent of the tool version and $3)$ unexpected behavior can come up from hidden implementation details of the simulator.

3 Formal Model

The formal model on which we base the correct-by-construction transformations presented in this paper is the *synchronous reactive model of computation* (SR MoC) [3]. It is a block-diagram formalism devised for specifying synchronous real-time systems, where system specifications are provided as an interconnection of blocks. The model of time is discrete and represented as a totally ordered sequence of "ticks" (logical time).

Blocks can have state and their outputs can also be combinatorial functions of the inputs. Interconnections represent signals, that are defined as functions over the sequence of ticks. Interconnections are regarded as zero-delay communication channels between blocks, unless one or more *memory elements* are introduced on the interconnection channel, each one specifying a one-tick-delay.

The execution semantics of the SR model is based on fixed-point computation, which is used to compute the stable point of the interconnected combinatorial functions at every tick. When no combinatorial loop exists the fixed-point computation reduces to causal computation: if some output of a block is interconnected to some input of another block, the former is required to execute before the latter.

The set of values that a signal can take up at each tick is extended with a special symbol $\perp$ called *bottom*. This way,

at each tick, all signals are assigned a value even though the signal is actually undefined for that tick (in this case, we assign the signal the $\perp$ value). SR blocks are allowed to execute only when at least one input is different from $\perp$.

The choice of the SR MoC is dictated by its expressiveness, i.e. on the basis of what the model allows us to represent and how "economic" is this representation. The SR MoC is a good match for control intensive systems, exactly the ones that are specified in ASCET and Simulink (at least for automotive applications). Particularly, the Simulink model of computation is based on discrete time simulation and it is fully compatible with the SR MoC, also discrete time.

To better understand how SR and ASCET MoC relate, we must dive in some more details into the TPM paradigm. In general, the TPM can be regarded as an "asynchronous" paradigm, in the sense that each task (or process) has internally no notion of synchronization with other tasks (or processes). This makes the behavior of the system specification dependent on implementation details such as execution time and preemption. In fact, depending on tasks' execution time, the presence of preemption may invert the order of process executions, which may differ from the causal order. Moreover, preemption effects may even impair the consistency of shared variables. This contrasts with the SR MoC where the "atomic" execution of each block prevents inputs to change during execution, and the behavior of the system at each tick is determined by the fixed-point semantics.

Nonetheless, well-known techniques can be implemented to enforce data consistency and a deterministic execution order of tasks as dictated by causality. Notice that, since tasks are executed once at each activation reference no combinatorial loop exists and the causal order ensure the convergence of the fixed point computation.

Our analysis focuses on single processor implementations as supported by ASCET. Since every single processor implementation is endowed with a global scheduler, we resort to scheduling techniques such as priority assignment and the priority ceiling protocol [8] for shared resources and we define the procedure in Figure 1 to check (and possibly enforce) whether the behavior of the system implementation is consistent with the SR semantics. In [7] a more conservative check is presented for fixed-priority scheduling with lock-free data sharing.

The procedure of Figure 1 is based on the analysis of data dependencies and priority assignment. When T_{EX} and T_{SEP} cannot be measured, we can assume the worst case given by $T_{EX} = \infty$ and $T_{SEP} = 0$. In such a case the algorithm only checks (and possibly enforce) that the priority assignment complies with the causal order, i.e. if T_1 sends a message to T_2, then T_2 must not be assigned a higher priority than T_1.

Figure 5 shows the task communication pattern, where

Def.1: let $P(A)$ be the priority assigned to task A.
Def.2: the duration of task A is a known upper bound of the execution time of A when it completes execution without being preempted.
Def.3: let $T_{EX}(A)$ be an upper bound of the execution time of task A. This is a function of the durations of the tasks that can preempt task A.
Def.4: let $T_{SEP}(A, B)$ be a lower bound of the time separation between any activation time of A and any activation time of B, when the former precedes the latter.
Step 1: build a dependence graph, where vertices represent tasks. Define a directed arc from vertex A to vertex B if task A modifies a protected resource accessed by task B.
Step 2: label every directed arc from A to B with $L_T = T_{SEP}(A, B) - T_{EX}(A)$ and with $L_P = P(A) - P(B)$.
Proposition 1: If tasks, whose activations can be simultaneous, either have different priorities (the execution order is fixed) or do not modify any common shared resources (the execution order does not matter), a sufficient condition for a single processor implementation to be deterministic is that for every arc at least one of L_T and L_P is non-negative.
Step 3: check that for every arc at least one of L_T and L_P is non-negative.

Figure 1. Data dependency analysis algorithm

M_i represents a protected shared variable, and the corresponding dependence graph. Since the arc from $T4$ to $T3$ has both a negative $L_P = 5-10$ and a negative $L_T = 5-7$, the implementation is not guaranteed to be deterministic, because preemption may invert the order of task executions with respect to the task activation order. After changing the priority of $T4$ to 10, L_P becomes 0 on the loop and determinism is ensured by the algorithm.

4 SR Interpretation of System Specifications

Every ASCET specification that is compliant with the rules given in Section 3 can be interpreted within the SR MoC. In fact, ASCET provides all the basic mechanisms, i.e. variable protection, priority assignment, priority ceiling, to ensure this compliance.

Let us start by looking at a simple example. An ASCET specification of a seat-belt alarm controller is divided into 5 modules, as shown in Figure 6, each consisting of a single process. The task specification comprises 5 different tasks to accommodate the 5 processes.

In ASCET the state of processes is represented by their local variables. Communication between processes occurs in ASCET via messages and shared variables. The ASCET interpretation of the seat-belt alarm controller example within the SR MoC is shown in Figure 7. The interpretation procedure is summarized by the rules in Figure 2.

Referring to Rule R1 and Figure 8, since processes are assigned to tasks and the latter are executed on a given activation reference, a "trigger" input signal is added to each block in the SR model. By trigger, we mean a signal that

R1: Each ASCET process is mapped to an SR block as shown in Figure 8.
R2: ASCET messages and shared variables become SR channels.
R3: Memory elements are introduced to preserve scheduling order defined in ASCET whenever a task sends a message to a higher-priority one.

Figure 2. ASCET to SR transformation rules

R1: SR blocks can be introduced as Simulink blocks with the same next-state and output functions,
R2: SR block interconnections correspond to Simulink interconnections.
R3: triggering inputs, if present, are mapped to triggers,
R4: Memory elements are mapped to Simulink continuous time memories.

Figure 3. SR to Simulink transformation rules

is defined only on ticks corresponding to the activation instants and takes up the value $\perp$ otherwise. While the SR block is executed at each tick, the role of the triggering signal, is to make the function defined by the ASCET process be executed only when the trigger value is not $\perp$ (the function is not executed when all its inputs are $\perp$). In all other cases, the SR blocks preserve the last computed values on its outputs.

As far as Rule R3 is concerned, the higher-priority task will be scheduled first using the *old* value of the message written by the lower-priority task. This is modeled in the SR MoC by the memory element that holds the *old* value on the channel.

A Simulink [1] specification can be interpreted according to the SR model in a simple way if at the top hierarchical level every loop between blocks is broken by at least one memory element. In this case no combinatorial loop can exist and the blocks can be considered as SR blocks, while the Simulink continuous time memory element can be transposed into an SR memory element. To interpret triggered blocks as SR blocks, the triggering signals must be considered as triggering inputs of the corresponding SR block.

An example of an SR interpretation of a Simulink specification is shown in Figures 10 and 11.

5 Model Transformation

5.1 Porting ASCET Projects to Simulink

The SR interpretation of Simulink specifications given in Section 4 shows that SR specifications that do not have combinatorial loops can be easily ported in Simulink by applying the rules in Figure 3.

Hence, any specification that can be interpreted with an SR model that do not exhibits combinatorial loops can be easily transformed into a Simulink model by deriving the SR model of the specification and applying the transformation rules of Figure 3. In particular, given the ASCET SR interpretation provided in Section 4, a correct by construction transformation from ASCET to Simulink is easily obtained by applying the Rules in Figure 2 and Figure 3 in sequence.

[1] We refer to Simulink V6.5 R14 version

R1: Every SR block is mapped to an ASCET process.
R2: Interconnections between processes are realized in ASCET as protected messages.
R3: ASCET processes corresponding to SR blocks with the same triggering input are organized within the same task in a sequence that is compatible with the causal order in the SR model.
R4: Triggering inputs become task activation references.
R5: Scheduling priorities are assigned so that if B causally depends on A, then either 1) A is assigned to a higher priority task than B, or 2) A and B have the same triggering input and are assigned to the same task with A preceding B.

Figure 4. SR to ASCET transformation rules

Figure 9 shows the Simulink specification obtained by the SR model of Figure 7, which in turn is the SR interpretation of the ASCET project shown in Figure 6.

5.2 Porting Simulink Specifications to ASCET

The SR interpretation of ASCET specifications given in Section 4 shows that SR specifications in which every block has one triggering input and no combinatorial loop exists can be easily ported in ASCET by applying the rules in Figure 4.

The priority assignment rule (R5) guarantees a sufficient condition for the ASCET specification to comply with the SR semantics. Observe that the algorithm of Figure 1 provides a less conservative condition for the priority assignment, that can be exploited when T_{EX} and T_{SEP} are known.

The above rules show that it is straightforward to define a rigorous transformation of a Simulink specification to ASCET when the SR interpretation of the Simulink specification is as introduced at the beginning of this section, i.e. at the top hierarchical level every block is triggered and every loop between blocks is broken by at least one memory element.

The condition above is not restrictive. Indeed, SR blocks without a triggering signal, as defined in Section 4, are executed at every tick, which can be considered as their implicit triggering input. From the TPM perspective this results in an inefficient implementation. Nonetheless, in many cases

used in practice functional analysis of the SR block allows to relax the triggering condition. Due to space limitations it is not possible to discuss exhaustively how functional analysis can be carried out. As an example, consider a block without triggering signals whose inputs all come from triggered blocks. If the block modifies its state and outputs only when its inputs change, then it is not necessary to execute the block at every tick, because the block can be equivalently executed on the triggering signals of its input blocks.

In Simulink a given triggered block can be activated by several signals that are multiplexed into a single trigger. If at a given simulation instant more than one triggering signal exhibits a triggering event and the block is not a Stateflow block, the triggered block is executed once. Since we transform Simulink triggering signals into triggering inputs and in the end into ASCET task activation references, we need to assume that triggering signals of non-Stateflow triggered blocks are never simultaneous. This is because in the TPM model, tasks are not aware of simultaneous tasks activations.

The transformation of Simulink specifications to the TPM model has been independently discussed in [7], where the event-based SCADE simulation system is the target environment. In such work, different techniques are used for preserving the synchronous semantics, that require a dedicated RTOS support.

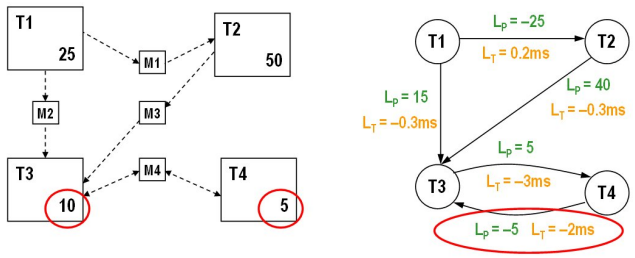
6 Conclusions

Using our method, model validation is not required each time models are transformed across different design environments. In fact, analysis and synthesis capabilities of the tool to which the model is exported can be applied. For example, a model imported from Simulink can be consistently integrated with other models specified in ASCET to produce an OSEK compliant application taking advantage of ASCET automatic integration features.

We believe that future design flows will be based on a similar approach. We envision that an intermediate format with formal semantics for embedded software models could be defined where the semantics of the different tools could be embedded and manipulated easily. The existence of this intermediate format with formal semantics does not guarantee that mapping a model from an environment to another is transparent. It will provide though an important bridge to map one semantic model into another avoiding the N^2 translation problem.

References

[1] P. Caspi, A. Curic, A. Maignan, C. Sofronis, and S. Tripakis. Translating discrete-time simulink to lustre. In *Proceedings of the International Conference on Embedded Software*, 2003.



Task	T1	T2	T3	T4
Period(ms)	1.0	1.0	10.0	10.0
Delay(ms)	0.0	0.5	0.0	5.0
Duration(ms)	0.3	0.3	2.0	1.0
T_{EX}(ms)	0.3	0.3	8.0	7.0
Priority	25	50	10	5

Figure 5. Data dependency analysis example

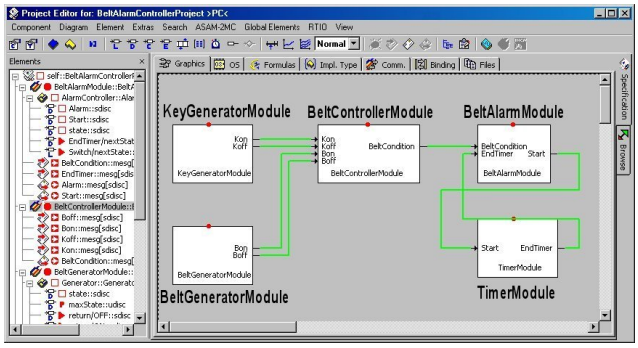


Figure 6. Top view of ASCET specs

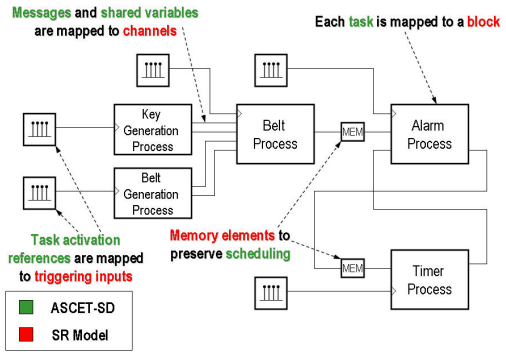


Figure 7. SR interpretation of ASCET specs

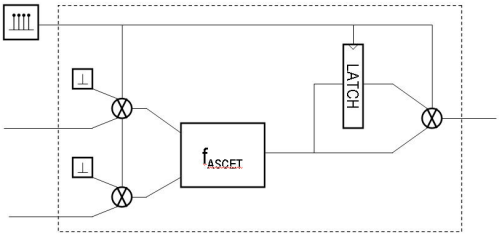


Figure 8. SR interpretation of ASCET processes

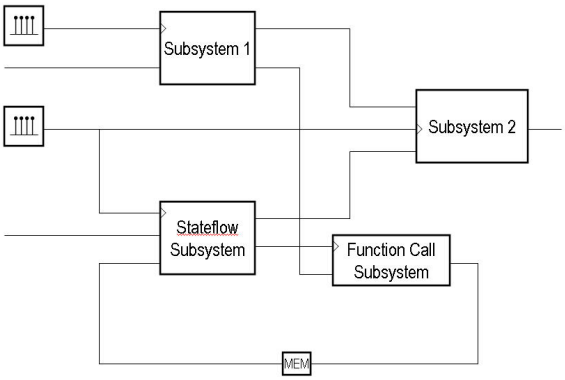


Figure 11. SR representation of Simulink specs

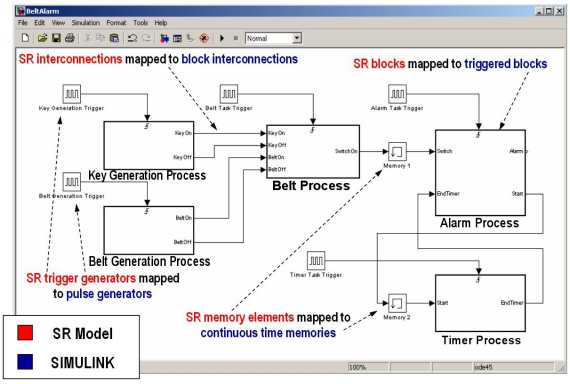


Figure 9. Simulink model equivalent to ASCET specs

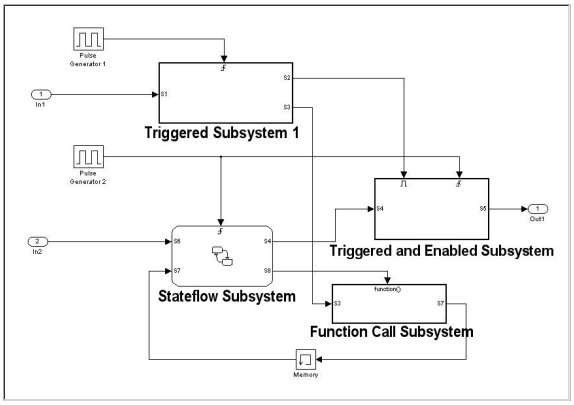


Figure 10. Original Simulink specs

[2] S. Edwards, L. Lavagno, E. Lee, and A. Sangiovanni-Vincentelli. Design of embedded systems: formal models, validation, and synthesis. *Proceedings of the IEEE*, 85(3):366–390, Mar. 1997.

[3] S. A. Edwards and E. A. Lee. The semantics and execution of a synchronous block-diagram language. *Science of Computer Programming*, 48(1):21–42, jul 2003.

[4] ETAS. ASCET. http://www.etas.de.

[5] A. Ferrari, G. Gaviani, G. Gentile, G. Stara, L. Romagnoli, and T. Thomsen. From conception to implementation: a model based design approach. In *IFAC Symposium on Advances in Automotive Control (IFAC-AAC'04)*, apr 2004.

[6] T. Henzinger, C. Kirsch, M. Sanvido, and W. Pree. From control models to real-time code using giotto. *IEEE Control Systems Magazine*, 23(1):50–64, January 2003.

[7] N. Scaife and P. Caspi. Integrating model-based design and preemptive scheduling in mixed time- and event-triggered systems. In *Proceedings of Euromicro Conference on Real-Time Systems (ECRTS'04)*, 2004.

[8] L. Sha, R. Rajkumar, and J. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, Sept. 1990.

[9] The Mathworks. MATLAB/Simulink. http://www.mathworks.com.