Enabling Model-Based Development of Distributed Embedded Systems on Open Source and Free Tools

M. Di Natale, M. Bambagini, M. Morelli
Scuola Superiore Sant'Anna, Italy

A. Passaro, D. Di Stefano, G. Arturi
Evidence Srl, Italy

Abstract

Model-Based Design brings the promise of an improved quality and productivity in the development of embedded systems and software. Flows based on commercial tools are today used in the industrial practice, albeit with several limitations. Furthermore, the analysis of the time properties considering scheduling and communication delays requires the addition of custom blocks to functional models, violating a desirable separation of concerns between the functional and the platform designs. Finally, to get full control at all stages in the modeling of systems and the automatic generation of implementations, it would be desirable that the toolchain is available as open source in all its components. We outline the initial steps in the development of a framework, largely based on the Scicos and Eclipse EMF open frameworks, that aims at providing support for the modeling, simulation, analysis and automatic generation of implementations of embedded functions on complex, distributed architectures.

1 Introduction

The use of models for the advance analysis of the system properties and verification by simulation, the documentation of the design decisions and possibly the automatic generation of the software implementation is an industrial reality, backed by several commercial products. The Model-Based Design approach (MBD) prescribes models based on a synchronous (reactive) execution model. Examples of available commercial tools are Simulink [5] and SCADE [7]. These tools are feature-rich and allow the modeling of continuous-, discrete-time, and also hybrid systems in which functionality is represented using an extended finite-state machine formalism.

However, MBD commercial tools lack in the capability of modeling physical computing architectures (and to some degree also task and resource architectures), as well as computation and communication delays that depend on the platform. Also, available commercial code generators typically provide implementations only for code to be deployed on a single CPU (exceptions are the Rubus Component Model and the accompanying tools [2]). AADL languages also supports execution platform modeling [3], but the tools originating from these languages seldom allow for simulation and automatic code generation (one example is [4]). To fill this gap, designers provide an implementation for a distributed execution platform by adding custom-developed communication code and performing the application partitioning by hand.

The analysis of computation and communication delays can be performed using the Truetime blockset in Simulink and the generation of platform-dependent code (including the task structure, the I/O and the communication code) can be obtained with custom blocks (for example, a well-known solution in the automotive domain is the RTI blockset from DSpace). However, both solutions create a model in which the functional solution is interspersed with platform-specific implementation blocks. If the implementation platform, or the task placement, or in general the task configuration is modified, the user must change all the affected blocks inside the model. Furthermore, code generation blocks for I/O and communication tend to be platform-specific.

Separation of the functional and platform models is advocated by many: examples from the academia are the Y-chart [15] and the Platform-based design [17] approaches. The OMG (a standardization organization) in its Model-Driven Architecture (MDA) [8] defines a three stage process in which a Platform-Independent Model or PIM is transformed in a Platform-Specific Model or PSM by means of a Platform Definition Model (PDM). Finally, the automotive industry AUTOSAR standard [1] defines a virtual integration environment for platform-independent software components and a separate model for the (distributed) execution architecture, later merged in a deployment stage (supported by tools). Unfortunately, the AUTOSAR metamodel is public but not open (use is only authorized to members of the consortium). Similarly, the Eclipse EMF-based Artop tool that provides the basic support for the AUTOSAR metamodel and its serialization is open but restricted for contributions and use to the consortium members. Most impor-

tant, AUTOSAR does not have any feature for modeling the behavior of the functions. Therefore, an exetrnal tool or the actual code is needed for functional modeling.

An open source or free framework can be constructed leveraging the following toolsets:

- Scicos [6] can be used for the functional modeling. The graphical language of Scicos is based on a synchronous semantics that is close to the one of Simulink. The toolset is open, offers a scripting language interface and a TCL-based graphical front-end. Unfortunately, Scicos currently does not support the modeling of Finite State Machine blocks. In addition, the code generation available for the dataflow part of Scicos has several limitation: it generates one function call for each block, does not provide support for all datatypes, and only generates single-task implementations for single rate models. In addition, the generated code is derived from the code used by the simulator with the overhead for all the hooks and data structures that are useful at simulation time but not needed on the target.

- The Platform model, as well as the model of the tasks and messages, can be generated using the Eclipse EMF modeling framework [11]. EMF offers the possibility of defining custom metamodels using its Ecore facility. Alternatively, several tools leveraged the EMF to build UML or SysML [9, 10] modeling environments. The most popular among them are Topcased and Papyrus. SysML modeling has the advantage of being conformant to a standard, with graphical editors and a number of additional tools for supporting the management of models. Unfortunately, SysML (and UML) are generic modeling languages, with little support for (embedded) domain-specific definitions. Therefore, a specialized profile is needed before it can be used for modeling embedded platforms and systems. Such a profile is currently available from the OMG, as MARTE (Modeling and Analysis of Real-Time and Embedded systems) [16]. However, MARTE is of difficult use, often cumbersome and lacking in the availability of concepts for modeling physical communication devices, as well as messages.

- A code implementation can be generated from EMF models (Ecore, or the UML/SysML models of tools that are based on Ecore) using model-to-text generators (of which model-to-code is a special case). Among them, the Acceleo toolset [12] is open, based on OMG standards, and allow to write code generator templates that query the model using statements in the standard OCL language.

The resulting methodology is a merger of the MBD and MDA paradigms. The overall tool flow is represented in Figure 1. The functional modeling, simulation and possibly verification of functional properties is performed using Simulink or Scicos. when simulink is used, the correcial code generators from Mathworks can be used to produce the behavioral code. Alyternatively, the functional model is exported in the Ecore XMl format and inported in the Eclipse EMF. Here, it can be extended with the platform and mapping models (in Ecore or using SysML) and the platform-dependent portions of the code produced. In the following sections we provide more details on the technologies and tools that have been developed for this purpose.

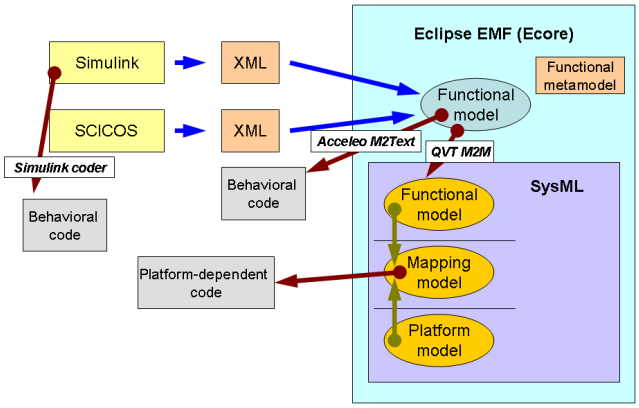

Figure 1. Exchange of information and code generation by the framework tools.

2 A Finite State Machine modeling tool for Scicos

To provide Scicos with the capability of modeling and simulating Finite State Machines, Evidence srl developed a modeling front-end, a simulation engine, a custom block which connects a Scicos model to the simulation engine, and a code genarator from a FSM specification.

The modeling tool (a snapshot is shown in Figure **??**) allows the specification of hierarchical (extended) Synchronous FSMs. Compared with commercial alternatives, or even UML/SysML State Diagrams, the model has several limitations. For example, no exchange of events among concurrent states, no join transitions, no inner or outer transitions and no access to internal variables from possibly concurrent states are allowed. This is done on purpose, to keep the FSM semantics concise and simple, thereby simplifying the code generation and possibly the verification of the behavior.

The tool produces an XML output describing the state machine structure and a stub for the creation of a custom Scicos block, representing the FMS in a larger model. An

executable reads the XML description and provides a simulation stub that can be connected to the Scicos simulator through the custom block to co-simulate the FSM in the context of the Scicos model.

2.1 Code generation from FSM blocks

A code generator produces an implementation of the FSM for execution on an embedded platform, consisting of an initialization function, and a step function to be executed at runtime (realizing the output update and the state update functions). The current generator produces an implementation based on a single step function. The code generator only accepts FSMs with periodic activation events and the step function that realizes the FSM behavior runs at the greatest common divisor of the activation events. However, in the future, we plan to extend the generator options to include a possibly more efficient partitioning of reactions in multiple functions to be scheduled at different periods [21].

3 Importing the functional model in the Eclipse EMF

The core of the modeling framework is implemented in the Eclipse EMF. Here, the functional model is matched with a platform model. A task, message and resource intermediate model is created in the process.

The functional model is created by importing in EMF the Scicos model (Alternatively, an input from Simulink is also possible). An Ecore metamodel has been defined for the import process (Figure 2)

This metamodel is not too dissimilar from the one proposed in the GeneAuto project [20] (actually, a simplified version of it). Contrary to GeneAuto, our metamodel is immediately available as an Ecore definition and it is only the starting point for a code generation process. GeneAuto has the objective of building an open-certified code generator, but does not handle distributed systems (and of course, neither the separation of functionality and task model) and does not generate platform-specific code.

In order to provide a code generator of industrial quality (with signal typing, code inlining and port variable folding) the functional model is then processed by a set of Acceleo code generation templates that produce the code that is functionally equivalent to the model.

At this stage, the generated code is strictly functional. Two generation modes are offered. In the first, a single-task implementation for the entire system is produced. The second mode allows for a multitask code implementation. When this mode is selected, we require that the designer partitions its model (at some level in the design hierarchy) in a set of superblocks (equivalent to Simulink subsystems)

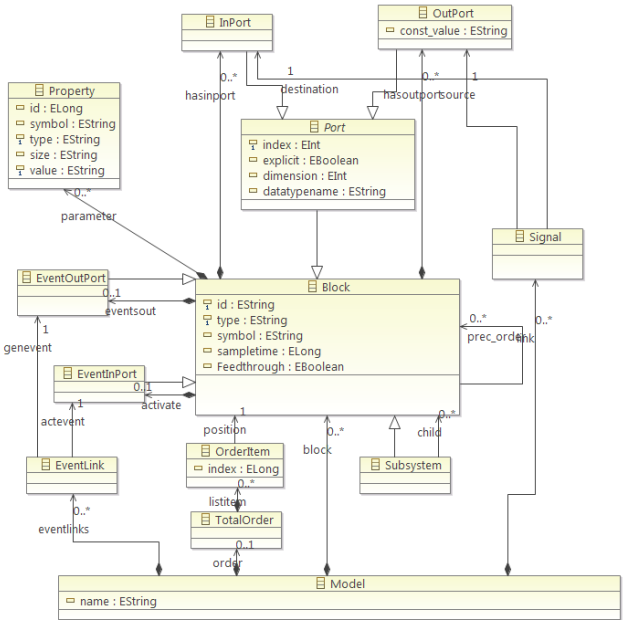

Figure 2. The Ecore metamodel for the functional part.

for which the execution is controlled by a single discrete-time clock. The superblocks are the units of execution for the code generation process. Each of the superblocks translates into a C function. Inside, the code implementing the blocks behavior (output update and state update) is generated and inlined, serialized according to the execution order generated by the Scicos modeler.

At runtime, access to the superblock ports does not happen in the way that is conventionally used by code generators. For example, the Simulink code generator project allows for several options, of which the most popular are to include the port variables in the signature of the Step function, or to use a set of global variables for the interface ports. Our generated code accesses the ports using a middleware-level API. Input or output ports are accessed using a simple and generic interface

```
EMW_read(sblock_id, port_id, port_type *read_var)
EMW_write(sblock_id, port_id, port_type write_expr)
```

This API eases the generation of the communication among superblocks in the case when these superblocks are mapped into different tasks or even remote nodes.

4 Matching the functional model with the platform model

Given that the functional model is produced by importing a description from Scicos, there is the need of a suitable

model (and metamodel) for the representation of the execution platform and the task and messages models. There are several options for the selection of such metamodels. In the following, we describe our current implementation, based on a set of custom metamodels, and then we outline another option, possibly more appealing for future work, in which the platforms and mapping models are generated by a customization of SysML.

4.1 Platform and mapping models based on custom Ecore metamodels

In our current implementation, the execution platform meta-model, shown in Figure 3, defines the hardware and software resources available in the system. The execution platform model needs to be modular and to support the concept of component libraries.

The main architectural elements that are used by the execution architecture designer are the following:

- *Embedded Control Unit* (ECU), which is a set of electronic boards, connected through communication links;
- *Board*, which may host several Controllers and Devices;
- *Controller*, which may include several Cores and Peripherals;
- *Core*, representing a computational unit;
- *Peripheral*, representing an electronic component which extends the Controller's functionalities;
- *Devices*, which represent I/O devices using a set of peripherals. So far, the meta-model supports buttons, touch screens, leds, lcd displays and servo motors. Communication units belong to a special class of devices, handled separately;
- *Real-Time Operating System* (RTOS), which may run on Cores.

The platform meta-model is completed by the project-specific definitions, showing the elements of the hardware and software architecture deployed for supporting the execution of the functions in one specific project instance. The main entities at this level are ECU instances (*ECUDeployment*), with the RTOSs executing on them (*RTOSDeployment*) and the communication buses (*Bus*). Each Bus object references the connected ECUs and the associated communication devices.

Figure 5 shows a screenshot of the developed Eclipse plug-in which defines the platform execution model out of the library components.

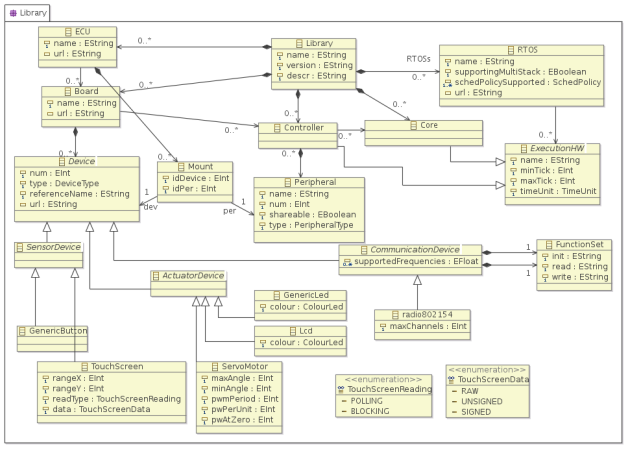

Figure 3. The meta-model for the library components that are used to construct the execution platform.

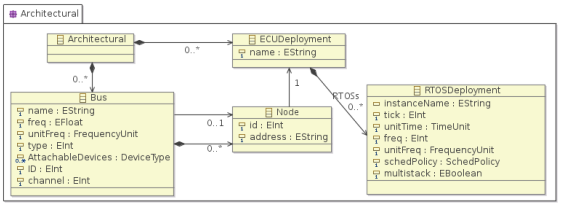

Figure 4. Meta-model of the objects that are used to build the project-specific execution architecture.

Once the system execution architecture is defined, a mapping model associates functional elements to tasks and then tasks to the (HW) processing elements, following the schema of Figure 6. The communication signals of the functional view are mapped (when needed) to messages and in turn, messages onto the physical links of the execution platform.

The task is the unit of concurrent execution that can run on one of the system cores, under the control of an operating system. The information about which RTOS hosts a specific task is handled by an association between the Task objects (Figure 6) and the RTOSDeployment (Figure 4). The Step methods of the functional subsystems are executed in the context of one of the task defined in the mapping model. More precisely, a list of Superblocks (defined as ProcMaps in the block-to-task mapping), sorted according to an execution order, belongs to each task. Each mapped Superblock refers to the appointed Step method. Moreover, designers must specify all the information concerning task times (such as period and deadline) and scheduling (priority and

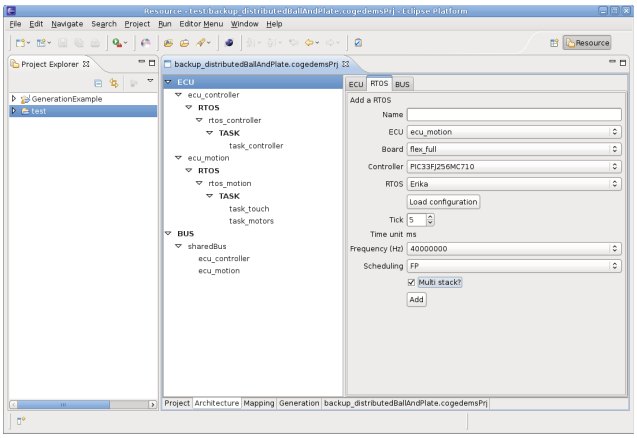

Figure 5. The editor used to construct the model of the execution platform.

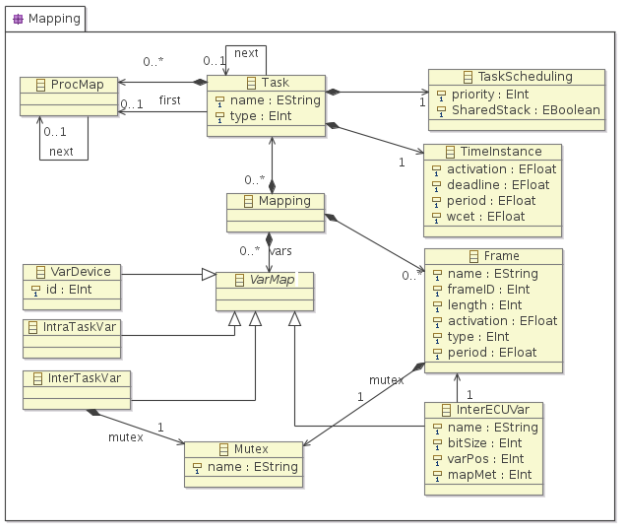

Figure 6. Meta-model for the definition of the mapping of functional components into tasks and connections among them.

reserved stack). The mapping of the functional subsystems (Procs) into tasks is subject to constraints and validation.

VarMap, the second important class into the Mapping package, concerns the mapping of signal variables or Vars (representing communication or I/O links, depending on the mapping). As shown in Figure 6, VarMap is a generic interface and four possible derived classes are instantiable: VarDevice is used to map custom devices to real devices and the other three to model all the possible communication scenarios according to the placement of the methods of Procs into tasks. *IntraTaskVar* communication takes place when two communicating procs are mapped into the same task (implemented using variables local to the task). *InterTaskVar* communication occurs between two tasks on the same Core. In this case, a suitable protection mechanisms for shared resources, provided by the operating system (part of the platform model) is used. The most complex case happens when the communication needs to be established between two remote tasks executing on different cores, or processors, connected by either an intra-chip network, a field bus or another network. An *InterECUVar* mapping object is automatically generated to represent this kind of communication, linking a Var to a specified portion of a *Frame*. Frames are periodic communication messages which are exchanged between two nodes through a shared bus.

4.2 Platform and mapping models based on SysML profiles and MARTE

A possibly more appealing option for the definition of the platform model and the mapping between functional subsystems and communication links and platform cores and communication or I/O interfaces consists in the use of a customized extension of the SysML metamodel. This allows us to use modeling tools with graphical editors, model checkers and processors, and possibly other standard tools. In this case, a stereotype of the standard SysML block concept could be used to represent Synchronous reactive subsystems and the standard SysML block definition diagrams (or bdd) and internal block diagrams (or ibd) could represent the block dependencies and the topology of communication. These models and diagrams, equivalent to the source Scicos or Simulink models, can be generated automatically starting from the Ecore model of the system functions, using a QVT transformation [19].

Next, the platform model could be constructed leveraging another set of SysML diagrams, using the MARTE stereotypes for computing nodes (cores), operating systems and computing resources, and suitable extending it for the model of the communication networks and protocols.

Finally, the mapping relationship can use the allocation tables of SysML, possibly complemented by an additional set of stereotypes to define the mapping of functional subsystems (their step functions) onto tasks and of signal data onto messages. Once again, MARTE provides standard stereotype concepts for representing tasks, shared resources and the attributes that are required by the most popular real-time analysis techniques.

5 Code generation and Example

The code generation of the platform-dependent code, including the tasking model and the communication code can be performed by processing the platform and mapping mod-

els using Acceleo (model-to-text) transformation templates. Acceleo is an open tool and allows great flexibility in the generation of the program implementation. In our current implementation, we are assuming an OSEK interface for the generation of the task model and the operating system-level API for task management and an AUTOSAR interface for I/O management. The tasks code is generated as a sequence of calls to the step methods of the functional subsystems mapped into it. Calls are sequentialized according to the mapping order, which must be consistent with the partial order of execution specified by the semantics of the functional model. Each invocation of the *step* method is preceded/followed by the respective middleware read/write functions implementing the functional data flows.

The code implementation of the middleware read/write functions, representing the communication links (*Vars*), depends on the task and core mapping of the blocks at the two endpoints of the communication.

For each communication signal, the Acceleo scripts can generate four different implementations. When the Var mapping is an instance of *IntraTaskVar*, the access functions execute read/write operations to a shared variable. If the endpoints are located on different tasks managed by the same operating system (*InterTaskVar*), read and write accesses to the shared variable are realized using a lock-free access protocol. The third case, which occurs when the sender and the receiver are on different processors (*InterECUVar*), is handled by using a middleware implementation wrapping a network communication API. Finally, communications between controller subsystems and the plant model can be mapped into I/O primitives.

A simple test case for the proposed methodology has been implemented by realizing a ball-and-plate controller with the same functional model (a PID controller) and two distributed implementations: one as a single node, and the other as a distributed platform (two nodes connected by a radio link). The two implementations have been obtained by changing the platform instance and the mapping specification and without writing a single line of code.

6 Conclusion

In this paper, we presented the overall architecture for a set of tools supporting a model-based development process for complex-distributed systems, from the system-level modeling of functions to the generated code

References

[1] *The AUTOSAR Standard, specification version 4.0*, the AUTOSAR consortium, web page: http://www.autosar.org.

[2] K. Hänninen, J.M. Turja, M. Nolin, M. Lindberg, J. Lundbäck, K.L. Lundbäck, The Rubus Component Model for Resource Constrained Real-Time Systems, 3rd IEEE International Symposium on Industrial Embedded Systems, Montpellier, France

[3] G. Raghav, S. Gopalswamy, K. Radhakrishnan, J. Hugues and J. Delange, Model based code generation for distributed embedded systems, European Congress on Embedded Real-Time Software, 19-21 May 2010, Toulouse, France.

[4] Hugues J., Zalila B., and Pautet L. Rapid prototyping of distributed real-time embedded systems using the aadl and ocarina. In 18th IEEE/IFIP International Workshop on Rapid System Prototyping (RSP'07), Brazil, 2007.

[5] "The Simulink product web page," http://www.mathworks.it/products/simulink/index.html.

[6] "The Scicos project web page," http://www.scicos.org/.

[7] "Scade product web page," http://www.esterel-technologies.com/products/scade-suite/.

[8] "The OMG Model Driven Architecture initiative," web page, http://www.omg.org/mda/.

[9] "The Unified Modeling Language (UML)," http://www.uml.org/.

[10] "The OMG Systems Modeling Language (SysML)," http://www.omgsysml.org/.

[11] "Eclipse Modeling Framework (emf)," http://www.eclipse.org/modeling/emf/.

[12] "The Acceleo model-to-text language and tool web page," http://www.acceleo.org/.

[13] "Real-time workshop/embedded coder," http://www.mathworks.com/products/matlab-coder/index.html.

[14] Y. Vanderperren and W. Dehaene, "From uml/sysml to matlab/simulink: current state and future perspectives," in *Proceedings of the conference on Design, automation and test in Europe: Proceedings*, ser. DATE '06.

[15] B. Kienhuis, E. F. Deprettere, P. v. d. Wolf, and K. A. Vissers, "A methodology to design programmable embedded systems - the y-chart approach," in *Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation - SAMOS*, 2002.

[16] "The OMG Marte Profile web page," http://www.omgmarte.org/.

[17] F. Balarin, L. Lavagno, C. Passerone, and Y. Watanabe, "Processes, interfaces and platforms. embedded software modeling in metropolis," in *Proceedings of the Second International Conference on Embedded Software*, ser. EMSOFT '02, 2002.

[18] "The eclipse modeling project," http://www.eclipse.org/modeling/.

[19] "The QVT transformation language," http://www.eclipse.org/qvt/.

[20] "The Gene-Auto project web site," http://geneauto.gforge.enseeiht.fr/.

[21] M. Di Natale, H. Zeng"Task Implementation of Synchronous Finite State Machines," DATE conference 2012.