Combined usage of UML and Simulink in the Design of Embedded Systems: Investigating Scenarios and Structural and Behavioral Mapping

Jianlin Shi¹, Martin Törngren¹, David Servat², Carl-Johan Sjöstedt¹
DeJiu Chen¹, Henrik Lönn³
KTH, Sweden¹, CEA List, France², Volvo Technology, Sweden³
jianlin/martin/chen/carlj@md.kth.se, David.Servat@cea.fr, henrik.lonn@volvo.com

Abstract

The multidisciplinary nature of advanced embedded systems requires a combined usage of several tools and modeling languages in systems development. We investigate the needs and some of the possibilities in simultaneous usage of Matlab/Simulink and UML. Structural and behavioral mappings are explored considering the needs for models at different abstraction level as well as environment models. The representation and mapping between behavioral models, including discrete-time, event-triggered, and continuous time systems is of special concern and solutions are discussed.

1. Introduction

There are several traditional ways by which developers try to manage the integration of embedded systems. We distinguish between hardware level, software level and model level integration [5]. Among these approaches, the model level integration has a major advantage in respect to both product qualities and engineering effectiveness by enabling earlier solution integration, verification and validation (V&V), and design space exploration [9].

The design of advanced embedded systems requires a consideration of several levels of abstractions, from requirements to the implementation in software and hardware, and also the environment with which the system interacts. Embedded system products are moreover characterized by the need of considering a multitude of conflicting qualities and aspects in their development. As a consequence, there is a need for combining the usage of several disciplinary tools and modeling languages.

In this paper we focus on the needs and possibilities concerning the combined usage – i.e., model level integration - of Matlab/Simulink [10] and the Unified Modeling Language (UML) [11]. This investigation is motivated by the fact that Matlab/Simulink is a very common tool for model-based control system design and in particularly simulation, whereas the design of embedded systems software architecture as well as the design of other functionalities is increasingly based on the evolving UML and its derivatives. The original purpose of UML is to provide graphical views of software systems. There is an increasing industrial interest in supporting the integration of Matlab/Simulink with UML. Several tool developers and research papers have begun to investigate and propose solutions.

In this paper we approach the combined usage of Matlab/Simulink and UML in two ways, by
- Identifying scenarios and providing motivation for model transformations.
- Investigating structural and behavioral mappings between subsets of the two modeling languages.

Both Matlab/Simulink and the UML are rich languages. For the study of mappings the investigation is therefore delimited not to include Stateflow behavior in Simulink. For UML we focus on the composite structure diagram and a subset of UML2 behaviors, mainly the activity diagram. The work is performed in the context of the ATESST project [1], where an architecture description language EAST-ADL2 for automotive embedded systems is developed and implemented as a UML2 profile. Structural mappings between Simulink and UML2 have been developed in the project. This paper reports on this work together with preliminary results from an investigation on why and how to deal with behavioral mappings between Simulink and UML.

The paper is structured as follows. Section 2 describes needs and example scenarios for integration, while in this section we also provide references to relate work in connection to relevant scenarios. Section 3 describes structural and behavioral aspects of UML and Simulink, and explores structural and behavior mappings. Section 4 concludes the paper with a discussion and openings for further work.
2. Integration needs

This section first provides overall requirements and context for embedded systems modeling, and then concretizes these with model integration scenarios.

2.1 Scope of models and dependencies

From the design point of view, two orthogonal dimensions of a system can be identified: system content and abstraction level. The content includes system structures and behaviors, representing “what the system is” and “what the system does” respectively. An embedded system typically includes behaviors of different types such as discrete-time, continuous-time, and discrete-event. The abstraction levels are defined according to the degrees of preciseness and detail with respect to the final realization, ranging from abstract function specifications to operational software code. Figure 1 illustrates four abstraction levels of an embedded system and the tight connection to the systems environment, which has to be considered at all levels. An engineering process has to deal with these levels and also covers V&V and information management.

![Figure 1 Abstraction levels of embedded systems](image)

Each abstraction level is populated by design entities, from abstract models to physical components, such as functions, software, and hardware components. There will obviously be dependencies among these components. Examples of such dependencies include [6]:

- Communication/composition at a particular level – e.g., between software components.
- Refinement – where e.g., a set of functions are refined into a piece of software.
- Allocation – e.g., where software components are allocated to processors.
- Duplication – e.g., due to undesired overlap of information among models.
- Commonality – e.g., because of reuse of functions.

2.2 Simulink/UML model integration scenarios

Given that UML and Simulink are used to represent different views/concerns related to the product, the above types of dependencies will be introduced between the models. This causes a general need for creating and managing these dependencies. Moreover, it is highly desirable to support dedicated system views as well as system level analysis and synthesis. Examples of the two latter include consistency checks across the models, system level quality analysis (e.g., safety analysis or co-simulation), and coordinated generation of code/documentation.

In the following we provide example scenarios which serve the main purpose to illustrate the needs for structural and behavioral transformations.

2.2.1 UML for systems engineering and Simulink for control function development

From the viewpoint of the ATEST project this is the main use case which in turn involves several scenarios. In this case, Simulink is considered as a specific system view, whereas the UML2 models contain the complete system model. Given that the functionality is first developed in Simulink, it would be desirable to import this into the UML2 models. Alternatively, given that the UML2 system model is first developed, the functional model could be exported to yield a Simulink model. As will be discussed in Section 3, the corresponding transformations can be done in different ways depending on the purpose and how the Simulink model is interpreted (as structure and/or behavior). One example of usage of the behavior models, would be to have UML2 activity diagrams to specify the complete system behavior, where a subset may correspond to the Simulink design. In this case, behavior transformations are of interest to establish.

2.2.2 Simulink for functions and UML for software design

A common case is where Simulink is used as a basis for control function development whereas UML2, e.g., in terms of class or composite structure diagrams, is used to design the software and hardware structure. All these structures are related to each other through refinement and allocation relationships. In some cases it is desirable to let one of the structures be dominating. Structural transformations can then be used to impose e.g., the functional structure as the basis for the software structure or vice versa. Another possible scenario is where it is desirable to analyze the software design through simulation. In the latter case, a transformation of behavior
aspects to Simulink from the UML model is required.

An example of an effort in this category is the mapping from the AUTOSAR\footnote{AUTOMotive Open System Architecture} software component description to Simulink models as a basis for behavior modeling and simulation \cite{2}.

2.2.3 Subsystem composition

It is common that functions/subsystems are developed concurrently by different engineering teams. For example, a discrete-event controller may be designed using UML diagrams and the environment in Simulink, or the feedback control system using Simulink, and other functionality using UML diagrams. In this case it may be desirable to generate a system view that encompasses both models. This could be accomplished by transferring one of the models, or a subset thereof, into the other tool. Co-simulation may also be desirable. Case tools like ARTiSAN Real-time Studio, I-Logix Rhapsody and Rational Rose Realtime claim support for co-simulation with Simulink. Such approaches to co-simulation are beyond the scope of this paper; they focus on run-time simulation and interaction between tools/models or on transferring one of the models in the form of software code to the other tool \cite{7}. However, this transfer may also be possible at the model level given that the models of computation are compatible. One example of model level integration is GeneralStore \cite{3}, which performs structural transformations between Simulink models and UML class diagrams, where the UML models are used as the overall system representation.

As discussed previously, UML and Simulink have different merits in terms of behavioral definition. Although model-level behavioral integration can be achieved by transforming behavior to a common representation, the original behavior may well be the best description. In light of this, the behavioral integration should be seen as a part of the system analysis. In this case, the differences and possibilities for mapping among behavioral descriptions in UML and Simulink have to be known.

3. Mapping Simulink and UML

Since both UML and Simulink are rich and semiformal languages and can be used in different ways, it is impossible to provide a “silver bullet” that gives a unique mapping. The mapping strategy varies depending on the actual needs. UML describes both behavioural and structural aspects, but all behaviors are attached to structural entities. However, Simulink essentially describes system behaviors, i.e., with a main emphasis on “what the system does” although a logical model structure can be imposed on the behaviors. This fundamental difference also means that different mapping strategies are possible. For example, the detailed design of a subsystem in Simulink could be mapped to UML structural diagrams for information management purposes or a behavioral description of one class representing the subsystem for behavior specification purposes. Another issue for the mapping is the interpretation of the languages vs. tool support and implementations. For UML, there are a number of profiles which extend the standard UML and a lot of UML tools with slightly different implementations of the UML. In this paper we focus on the UML standard although the option to explore profiling to enable certain mappings is discussed. This section in the following discusses structure and behavior in UML and Simulink, and explores solutions for their mapping.

3.1 Structural concept mapping

From a systems engineering viewpoint, the system structure represents concrete solution elements to which system functions are allocated. In modeling, a different logical structure can also be imposed on both behavioral and system structural abstractions. This logical structure, which has the purpose to facilitate model usage, may thus have no direct correspondence to the physical structure of a system.

3.1.1 Simulink model structure

Two types of composition can be used in Simulink model: virtual subsystems and atomic subsystems. Virtual subsystems are used for imposing a logical structure of a model. The boundaries of virtual subsystems are ignored in simulation and the semantics of the virtual subsystem structure relies on the user. The interpretation could for example be related to the physical system structure or the software structure. As a contrast, an atomic subsystem is treated as a unit when determining the execution order of blocks.

The basic elements of Simulink model are blocks and lines. A block represents a model entity, which might be a system that contains subsystems. To create a subsystem, it is necessary to include two kinds of special blocks Inport and Outport to represent the input-ports and output-ports of the subsystem. A line connects two blocks by connecting SrcPort and DstPort which correspond to the output-port and the
input-port respectively to represent the interaction of model entities in terms of data and control flows.

### 3.1.2 UML model structure

UML uses structural diagrams to describe the static relationships between system elements independent of time, where basic elements include *classes* and *associations*. Composition and aggregation are used to describe the part-whole relationship. UML2 has six kinds of structural diagrams, the class, object, composite structure, package, component, and deployment diagram. We focus on the composite structure diagram in this work. One reason for this is that the composite structure diagram was introduced into UML with the explicit purpose to show logical relationships between elements of a complex class and to identify their collaborations [8].

### 3.1.3 Structural mapping

In our approach, one assumption in the structural mapping is not to distinguish virtual subsystems from non-virtual subsystems. The modeling entities in a Simulink model can be treated as objects in UML, where in a general sense everything is defined by a class. *Signals* between Simulink blocks correspond to UML *connectors* with *ports* attached to the corresponding classes. Table 1 gives the proposed structural concept mapping in our approach. By this mapping, structures of a UML model depicted by connected classes through connectors, corresponds to structures described by connected blocks in a Simulink model. An example is given in Figure 2 corresponding to the analysis level of Figure 1. Table 1 does not show the details of dealing with different types of blocks, e.g., *Inport* and *Gain*, which describe structure and behavior respectively. In structural mapping, they are both described by UML classes but with different properties. The detailed information of functions can be described as connected classes in a composite structure diagram or as behaviors. This latter approach will be discussed in section 3.2.

#### Table 1: Structural concept Mapping

<table>
<thead>
<tr>
<th>Simulink Concept</th>
<th>UML 2 Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primitive block</td>
<td>Class</td>
</tr>
<tr>
<td>Subsystem</td>
<td>Property with a class defining it</td>
</tr>
<tr>
<td>Line</td>
<td>Connector</td>
</tr>
<tr>
<td>SrcPort</td>
<td>Port with a class defining provided interfaces for the port</td>
</tr>
<tr>
<td>DstPort</td>
<td>Port with a class defining required interfaces for the port</td>
</tr>
<tr>
<td>Line Branch</td>
<td>Connector</td>
</tr>
</tbody>
</table>

### 3.2 Behavioral concept mapping

The behavioral aspects refer to different models computation and communications (MoCCs). Basic elements to describe behaviors includes data, triggers, actions, states and time.

There are many types of MoCCs based on various combinations of these basic elements. For instance, simple MoCCs can be based on pure event-triggering (control flows), pure data-flow, or combinations thereof like client-server. Finite state machine formalisms (FSM) provide ways to express more complex MoCCs. Table 2 shows different MoCCs and examples of support from the UML and Simulink. It should be noted that although UML defines the concepts of *duration* and *time expressions*, there is no explicit concept of time or clocks as in Simulink. Detailed definitions of time including timers and clocks are given by UML profiles and specific tools.

#### Table 2: MoCCs and examples of support

<table>
<thead>
<tr>
<th>MoCCs</th>
<th>Simulink</th>
<th>UML2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client-server</td>
<td>No direct support</td>
<td>Sequence</td>
</tr>
<tr>
<td>FSM</td>
<td>Stateflow</td>
<td>State machine</td>
</tr>
<tr>
<td>Event-triggered</td>
<td>Conditionally exe-</td>
<td>Activity</td>
</tr>
<tr>
<td>and data-flow</td>
<td>cuted subsystem</td>
<td></td>
</tr>
<tr>
<td></td>
<td>and datal-flow</td>
<td></td>
</tr>
<tr>
<td>Discrete-time</td>
<td>Sampled blocks</td>
<td>Activity</td>
</tr>
<tr>
<td>Continuous-time</td>
<td>Continuous-time-</td>
<td>Activity</td>
</tr>
<tr>
<td>blocks</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2 Only examples of possible choices of diagrams are given here. See detailed discussions in the remainder of this section.
In the following we first briefly describe the MoCCs that are supported by Simulink and UML, and then discuss mapping and the representation of certain MoCCs in more detail.

### 3.2.1 Simulink model behavior

Three overall types of MoCCs can be modeled by Simulink: continuous-time, discrete-time, and event-triggered, with the exception of the client-server type which is not explicitly supported. Simulink uses different types of blocks, e.g., continuous-time blocks and discrete-time blocks for the corresponding MoCCs, as shown in Table 2. Primitive blocks are pre-defined in the Simulink library. Compositions of primitive blocks form subsystems which are virtual or atomic.

The behavior in Simulink can be interpreted in two ways: either from the language viewpoint or from the viewpoint of the observable simulation behavior. The simulation behaviors of a Simulink model depend on the precise ordering of the blocks during simulation. This ordering is determined by the simulation engine and depends among other things on the MoCCs involved.

### 3.2.2 UML model behavior

The fundamental unit for behavioral description in UML is the action. UML predefines a number of primitive actions such as the reading of and writing to properties of a classifier, sending signals, calling operations and/or behaviors. The concept of OpaqueBehavior with attributes (body, language) is introduced for user-defined behaviors given in terms of a specific language (e.g., C-code, text, reference to an external file, or an equation). The body attribute is a set of strings using this language to specify the behavior. UML uses data-flows to express object transfer, control-flows for the coordination between actions, and activities to express the composition of actions.

Based on the basic actions, there are three kinds of behavior formalisms in UML: activities, state machines and interactions. A state machine may have sub-machines; and an activity may be composed by sub-activities which are compositions of actions. Behaviors described by these formalisms are essentially restricted to discrete-time and event-triggered MoCCs. Figure 3 and Figure 4 give examples for each of them. The UML triggering mechanism can be combined e.g., event-trigger on some request and a periodic activation of an event. However, the interval between two events is not specified by the UML and can be as small as needed, and thus used for describing continuous behaviors. A continuous behavior can be specified by an UML OpaqueBehavior, which can be described by an activity diagram or be specified as a link towards an external Simulink file (by means of another OpaqueBehavior). The former approach is further discussed in section 3.2.3.

### 3.2.3 Behavioral mapping

This section introduces proposed behavioral mapping between UML and Simulink, starting with basic elements, i.e., primitive blocks and actions, and continues by discussing the mapping considering different MoCCs.

Since mathematical relationships described in Simulink are mainly data-flow and control-flow, it is natural to map the Simulink behavior to UML activity diagrams, which are described by connected executable nodes (action nodes), control nodes and object nodes. The use of activity diagrams for mathematical equations has been explored in [4]. Table 3 gives the behavioral concept mapping with four examples (inport, constant, sum and switch).

---

**Table 3: Behavioral concept mapping**

<table>
<thead>
<tr>
<th>Simulink</th>
<th>UML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primitive block</td>
<td>ActivityNode</td>
</tr>
<tr>
<td>Line</td>
<td>ControlFlow/DataFlow</td>
</tr>
<tr>
<td>Inport</td>
<td>ActivityParameterNode (ObjectNode)</td>
</tr>
<tr>
<td>Constant</td>
<td>ValueSpecificationAction (Action)</td>
</tr>
<tr>
<td>Sum</td>
<td>ActionNode</td>
</tr>
<tr>
<td>Switch</td>
<td>DecisionNode (ControlNode)</td>
</tr>
</tbody>
</table>

**Primitive blocks and actions**

Figure 5 shows a Simulink model and a corresponding UML model of the expression

\[
\dot{x} = 2x + u(t)
\]

(Eq. 1)

where blocks such as Sum and Integrator are represented by actions, e.g., Add and Integrate over time. A proposed strategy is to define such actions as part of a ModelLibrary of a UML profile, i.e., a dedicated package where types are defined, which are available when later applying the profile to a model. To do so, one could make a list of them in terms of OpaqueAc-
tions referencing functions available in another language, e.g., Simulink, via a pair of (body, language) attributes. If needed, more elaborated actions could be defined, made out of sequence of calls to such Simulink-based OpaqueActions and other calls. There is another downgraded version of this approach, which consists in defining dedicated stereotypes applicable to actions which would provide the meaning of Simulink functions – add, integrator, etc. – to already available actions. We propose the former approach because it provides an ability to instantiate actions e.g., a multiplier action – whereas the latter provides tags which do not have any interesting information except for their names.

![Image](image1.png)

**Figure 5** Simulink model and UML activity diagram for numerical solution of Eq. 1.

The mapping shown in Fig. 5 can be useful for system representation purposes. It is not however always adequate to represent the intended semantics in terms of the ordering of the actions. This is because the graphical view of the Simulink model in Fig. 5 may not represent the exact execution ordering. In Simulink many blocks contain states, e.g., the integrator in Fig. 5. The simulation behavior in the above Simulink model will in fact start at the output of the integrator, taking into account its initial value, and then continue with the Gain and Sum blocks. In the activity diagram of Fig. 5, the Add action will instead be provided with a default value and run before the Integrate over time. The difference in execution order may lead to different simulation results. This aspect reflects the fact that the mapping strategy has to take the purpose of mapping into account, such as for system behavior specification or detailed simulation behavior description. Indeed one could refine the first UML activity diagram in several versions to account for the different solver strategies used by Simulink.

Given the above mapping, a subsystem containing connected blocks in Simulink can map to an activity that is a composition of actions, and vice versa.

**Data types**

Simulink does not require an explicit declaration of types. Some blocks, e.g., Unit Delay and Sum, do not have type constraints on the block itself, instead they inherit the type from their input. However, Simulink does reject models due to type errors, e.g., when two inputs of a Sum block have different types when the setting of the Sum block is set to require the same type of inputs. Nine kinds of data types are defined in Simulink, known as boolean, double, single, int8, uint8, int16, uint16, int32, and uint32 respectively.

UML has four predefined data types, Boolean, Integer, UnlimitedNatural and String, known as instances of primitive type. In UML, a data type is similar to a class, but it is a special classifier whose instances are identified only by the value. Thus a user can simply define new data types when needed similar to defining a class. The mapping of data types between Simulink to UML therefore becomes a rather straightforward one to one mapping.

**Client-server:** Simulink does not have direct support for this MoCC. One solution to define it in Simulink is to use Stateflow charts (or masked S-functions) to model behaviors of the client and the server. The connection between the client and the server can be modeled by two triggers combined with data-flow. In the UML, the receiver could be described as a BehavioredClassifier, which corresponds to the client or server in Simulink. The behavior of the receiver is described by operations of the BehavioredClassifier. Corresponding to Stateflow charts used in Simulink, an execution protocol if any can be described by a state-machine and the exact computation being done within each operation can be described in an activity diagram.

**Event-triggered:** For this MoCC, a triggered block (conditionally executed subsystem) corresponds to a UML class which has a ClassifierBehavior attached to it. This class owns a behavior port which can have a Boolean attribute isBehavior set to true, meaning that requests arriving on this port are automatically triggering the ClassifierBehavior. Operation hosted by the classifier can be described in an activity diagram which corresponds to the behavior of the conditionally executed subsystem in Simulink.

**Data-flow:** For this simple MoCC, Simulink blocks can be mapped to UML ObjectNodes and vice versa.

**Discrete-time:** Similar to the event-trigger, a periodically executed subsystem can correspond to a UML class whose behavior is described by a classifier behavior. Instead of using behavior ports, a trig-
ger is linked to a TimeEvent. By using this approach, discrete time Simulink blocks can be described.

**Continuous-time:** A continuous-time model in Simulink can map to a UML model which has an OpaqueBehavior as its behavior specification. For example, a high level mathematical description corresponds to a UML OpaqueBehavior with (equation, math) assigned to its attribute (body, language). This OpaqueBehavior can be considered as a composite behavior like an atomic subsystem in Simulink. Its detailed behaviors can be depicted by an activity diagram. The mapping between Simulink block diagrams and UML activity diagrams has already been discussed earlier with examples shown in Figure 5.

### 4. Conclusion and future work

The development of complex embedded systems requires numerous specific views. We have studied the integration between Matlab/Simulink and UML by investigating integration scenarios and by exploring possible solutions for structural and behavioral mapping. Performing a mapping first requires consideration of the purpose and scope for the mapping in order to determine a suitable mapping strategy. While Simulink emphasizes behavior, it also includes logical model structuring, and a Simulink model can be interpreted at the language or the simulation behavior level. UML on the other hand makes a distinction between structure and behavior. But the language comes in several forms, including its definition, profiles and tool interpretations. For the behavior mapping, Simulink lacks explicit means to describe client-server behavior, whereas continuous-time modeling is not straightforward in the UML.

In our proposal, a restricted behavioral mapping can be defined between Simulink and UML2. For example, physical components modeled in Simulink can be imported to become a part of a system information model in UML, to support versioning, traceability to requirements, etc. We have also shown it is possible to map behaviors in terms of discrete-time, event-triggered and continuous-time MoCCs. In the particular case of the EAST-ADL in the ATESST project, behavior from different notations and legacy tools can be integrated, while keeping the integrated behavioral model sound. A prototype tool for structural mapping has been developed and the behavior transformation is under implementation.

### References


