Integrating SysML with Simulink using Open-Source Model Transformations

Andrea Sindico
RASD dpt.
Elettronica S.p.A.
Rome, Italy
andrea.sindico@elt.it

Marco Di Natale
ReTis Lab. TECIP Center
Scuola Superiore S. Anna
Pisa, Italy
marco.dinatale@sssup.it

Gianpiero Panci
RASD dpt.
Elettronica S.p.A.
Rome, Italy
Gianpiero.panci@elt.it

Abstract— **The realization of an integrated and automated modelling flow and tool framework joining OMG (i.e. UML, SysML, etc.) and Mathworks (i.e. Matlab, Simulink, etc.) models and technologies is a very attractive perspective because of the possibility of complementing their capabilities and strenghts. In this paper we describe our project for an integrated flow and our initial results, consisting of a transformation workflow to automatically generate Simulink models from SysML models and viceversa. Our proposed process is compliant with the OMG Model Driven Architecture. The transformations have been realized using the TopCased open-source modeling tool the Acceleo model-to-text generator.**

Keywords: SysML, Simulink, Model Transformation, MDA

I. INTRODUCTION

Model Driven Engineering [1] is a term encompassing all system engineering approaches focused on models definition and exploitation. These approaches are, in principle, meant to increase productivity by:

- maximizing compatibility among systems (via reuse of standardized models);
- simplifying the process of design (via models of recurring design patterns); and
- promoting communication among individuals and teams working on the system (via a standardization of the terminology and *best practices*).

In agreement with most modern development methodologies [1] [26] [24], MDE recognizes system development as a staged effort, in which a set of required functions, defined in an abstract or Platform Independent Model (PIM) are deployed, possibly automatically, onto an executable architecture. The result of this deployment is a Platform Specific Model (PSM). PIM models are closer to specifications and can be used for system-level functional decomposition and the abstract verification of functional properties. PSM models account for resource usage and can be used for quantitative analysis of performance, resource usage verification and planning.

PSM can finally represent the source for the possible automatic generation of a code implementation, documentation and other valuable artifacts.

Despite general consensus on the MDE objectives and continuous effort by the OMG in the development of modeling standards and by tool vendors in the support of the languages, quite often models are only used as a pictorial representation of requirements or functionalities. Verification or, in general, analysis of properties and especially automatic generation of implementations are far from being widespread. At now we can identify two major trends in MDE: Model Based Design (MBD) [24] and Model Driven Architecture [26]. Although they share objectives and principles they also present differences in the adopted approaches and technologies so that they are eventually considered competing or alternative. The Model Driven Architecture (MDA) [26] is an OMG [7] standard aimed at defining a reference architecture for a design and development process strongly based on the construction and analysis of models.

The term Model-Based Design (MBD) [24] instead indicates a slightly different approach and especially a different set of models and tools. While MDA originated from the move of a fundamentally software-oriented community (object-oriented design) towards system-level modeling and embedded systems development, MBD is very popular in the development of control-oriented functions and originated from the domain of control engineering and systems engineering. As such, MBD languages are usually based on a restricted but formal syntax and semantics, with an underlying Model of Computation (MoC) based on mathematical rules. A Synchronous Reactive semantics is the foundation of the most popular tools such as Simulink [19] and SCADE [25]. MBD models are executable and can be simulated. Verification of properties is made possible by the use of a formal MoC and a path to the automated generation of code is not only possible, but has become common practice in the automotive and aeronautics industry.

Even stricter is the selection of the platforms into which the Synchronous Reactive models of the MDB approach are to be deployed. The selection of the execution platform scheduling model and the software implementation are seldom (if ever) under the designer's control. Most commercial code generators allow for a single-core implementation of the models only (or at best, strict time-triggered approaches).

As a result, the two modeling frameworks tend to appeal to people with a different technical background. System analysts, System architects and software developers are typically more familiar with modeling languages such as

SysML [17] or UML, which are part of the MDA approach. Hardware, Firmware or Control engineers typically prefer the executable models of Matlab [18][20] and Simulink [19], with their capabilities for simulation, code generation and test coverage analysis, which are instead part of the MBD approach. Indeed, the two approaches might be combined to leverage their strengths. SysML modeling may be used at the architecture-level to define the system decomposition, the communication among subsystems, the execution platform, including the computation devices and the communication media that are available, and the deployment of the functional model into the execution architecture, which requires the creation of an intermediate software architecture model exposing and detailing the task and message architecture. Simulink models are to be preferred for the modeling of the internal behavior of blocks and for the maturity of the associated code generators when producing the software or firmware implementation of a complex function. Indeed, it is commonly accepted [2][3][4][5] that future trends in model engineering will encompass the definition of integrated design flows exploiting complementarities between UML or SysML and Matlab. The combination of the two models requires the capability of model-to-model transformations and integration of heterogeneous models. These operations are today often performed by hand, motivated by the fact that proprietary modeling languages, such as Simulink, lack a publicly available meta-model [2]. However, manual transformations should be avoided whenever possible since:

- errors could be introduced during the transformation process;
- the target model could not actually conform to the source model because of subjective erroneous interpretations of the source model.

A.   Proposed Framework

Our proposed framework is summarized in Figure 1. For simplicity, we describe it assuming a pure top-down development, although this is not required, since the framework supports also a bottom-up approach. The system is initially developed and partitioned in its major functional subsystems and components as a SysML model. Similarly, SysML supports the definition of the model of the execution architecture.
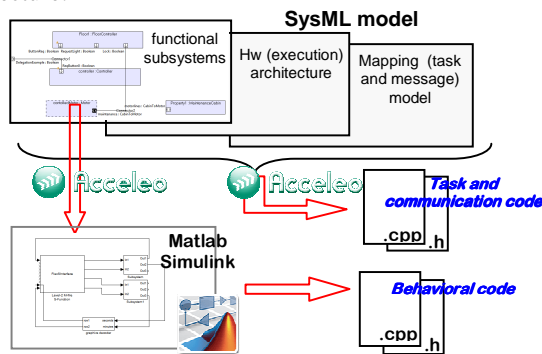

Figure 1.   : A Transformation Workflow integrating SysML with Simulink

The functional model includes the definition of the main functional subsystems (or SysML blocks) and their communication through ports. Subsystems are decomposed until they get to the level of units of deployment (that can be atomically allocated to computation nodes). Some of the SysML functional blocks can then be identified as executing according to a SR semantics (by means of suitable stereotypes). The execution architecture includes the model of all HW nodes, including firmware targets (FPGAs) and computing cores, of the networks and other communication buses/links and of shared memories.

Finally, a mapping layer completes the definition of the Platform Specific Model. Its aim is setting up an allocation relationships:

- o   between functional subsystems and computation cores
- o   between communication flows and messages
- o   between messages and networks in the case of distributed communication,
- o   between communication flows and shared memory locations in the case of software to firmware or intercore communications.

The behavior of some of the functional subsystems may be critical for the functioning of the system or their development may benefit for the use of simulation and verification capabilities. For these subsystems Simulink models will be developed using model-to-model transformations to retain the consistency of the subsystem interface specification. These Matlab/Simulink models can be executed and therefore validated against the available test cases (possibly automatically generated). In the development of the internal behavior, it is possible that the need to update the subsystem interface arises, generating a design iteration with a required change to the higher-level SysML model.

Code is generated from the models exploiting their capabilities and their information content. Behavioral code is generated from Matlab/Simulink models using the Real-Time Workshop/Embedded Matlab Coder suite, given that behavioral models are produced and validated in this toolset. The code implementing the tasks, as well as the communication (intertask, intratask, but also remote) code and part of the code implementing software-to-firmware communications, is generated from SysML models, given that this is the place where knowledge of the physical architecture and of the mapping of the functional subsystems into tasks and computing nodes is defined.

The code generation part of the project is still in progress. In this work we are documenting the model-to-model transformation technologies and the templates we used.

B.   State of the Art and Contributions

The match of a functional and execution architecture is advocated by many in the academic community (examples are the Y-cycle [27] and the Platform-Based Design PBD [28]) and in the industrial domain (the AUTOSAR automotive standard is probably the most relevant recent example [29]) as a way of obtaining modularity and separation of concerns between a functional specification and its implementation on a target platform. The OMG with

the MDA has similarly proposed a staged development in which a PIM is transformed into a PSM (although without an explicit separation between the execution architecture modeling and the mapping of function to architecture). Examples of methods tools and case studies in the application of the MDE transformation of a PIM into a PSM are ([32]). In reality, however, very few examples exist for the application of the proposed methodology to the mapping of complex functionality to a distributed system, in which the messaging and task structure arise because of the mapping. Similarly, the generation of code from a PSM is mostly limited to customization to a specific operating system or communication (middleware) layers.

As for the model-to-model transformations and heterogenous models integration, several approaches, methods, tools and case studies have been proposed. Several approaches (examples are the GME [30] and Metropolis [31]) propose the use of a general metamodel as an intermediate target for the integration of models. Also, the Eclipse modeling platform provides support for metamodel specifications through its Ecore (Eclipse-core) meta-metamodeling language. Model-to-model transformation engines are available for the Eclipse environment including ATL and QVT [14].

In this paper, we describe the initial results in the development of the proposed flow, consisting of an automatic workflow for SysML-to-Simulink models transformation.

Transformations can be used to automatically generate Simulink subsystem models from a SysML model in a top-down flow or to generate a SysML model of a Simulink subsystem in a bottom-up flow.

The proposed workflow is fully compliant with the OMG standards [7]. For its realization we used the Topcased [15] open source modeling environment based on the Eclipse Modeling Framework (EMF) [14].

The benefits of an automatic SysML to Simulink transformation are:
- the obtained Simulink models are necessarily conformant to the SysML models from which they derive;
- no time (and no effort) is needed to obtain Simulink models when the system SysML model is available;

II. THE MODEL DRIVEN ARCHITECTURE

The Model Driven Architecture (MDA) [26] is an OMG [7] standard aimed at defining a reference architecture for a design and development process strongly based on the construction and analysis of models.

To this end, the MDA is centred on the Meta Object Facility (MOF)[8], a meta-meta modeling language (a standard language to build meta-models) at the top of a four layers hierarchy as depicted in Figure 2. The next level (M2-models) includes the UML or SysML meta-models, that is, models that describe UML or SysML [9] themselves. M2-models describe elements of the M1-layer, which are, for example, models of a specific system written in SysML. Finally, the M0-layer or data layer is used to describe real-world objects.

The MDA is *model-driven* because it provides a means for using models to direct the course of understanding, design, construction, deployment, operation, maintenance and modification.

The most important means to manage models are *transformations*, defined in [11] as "*the automatic generation of a target model from a source model, according to a transformation definition*", which is, in turn: "*a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language*".
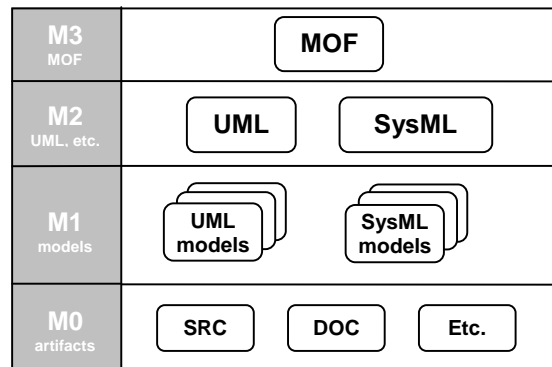

Figure 2. : The Model Driven Architecture Layers

In [11] Mens et al. also provide a taxonomy for model transformations according to which a transformation can be:
- *Exogenous*: when the source and target models conform to the same meta-model;
- *Endogenous*: when the source and target models do not conform to the same meta-model;

and
- *Vertical*: when the source and target models reside at different abstraction levels.
- *Horizontal*: when the source and target models reside at the same abstraction level.

The OMG provides standard languages for model-to-model transformations, such as the Query View Transformation (QVT) [12] language, and model to text transformations, such as the MOF to Text transformation Language [13]. A widely adopted CASE tool enabling a model-driven workflow compliant to the MDA standards is the Eclipse Modeling Project (EMP) [14].

EMP provides an implementation of the MOF language called Ecore. Ecore allows to define M2-models from which editors for M1-models can be generated. Then, models can be graphically defined and saved in the standard XMI format [rif]. Moreover, the EMP provides:
- an Eclipse plugin implementing the QVT language to define transformations between models possibly conformant to different meta-models (expressed in ECORE);
- an Eclipse plugin (called Acceleo [16]) implementing the MOF to Text Transformation Language which allows engineers to define

transformations from models conform to a specified meta-model (expressed in ECORE) and text.

The examples presented in this paper have been realized using TopCased [15], an open source extension of the Eclipse Modeling Project providing an ECORE implementation of the SysML profile and related editors.

III.   SysML and Simulink

The Systems Modeling Language (SysML) [10][17] is a general-purpose modeling language for systems engineering applications. It supports the specification, analysis, design, verification and validation of a broad range of systems and systems-of-systems. It is defined as an extension of a subset of the Unified Modeling Language (UML).

The major structural extension in SysML is the «block» which extends the UML Structured Class [17]. It is a general purpose hierarchical structuring mechanism that abstracts away much of the software-specific detail implicit in UML.

Blocks can represent any level of the system hierarchy including the top level system, a subsystem, or logical or physical component of a system or environment. Moreover, SysML blocks can represents hardware, firmware or software components, their parts, their interfaces and the data (signals) transferred amongst them.

An OMG SysML block describes a system as a collection of parts and connections between them that enable communication, data transfer and other forms of interaction. Ports provide interaction points for the communication and synchronization among blocks, when they are used within the context of a larger structure. SysML provides *Standard Ports*, which support client-server communication (e.g., required and provided interfaces) and *FlowPorts* that define flows in or out of a block (a signal or data interface).

Block Definition Diagrams (bdd) are used to describe the blocks, with their port interface and their internal attributes operations, parts and constraints. Also, a bdd diagram defines the relationships that exist between blocks. *Internal Block Diagram*s (ibd) instead are used to provide a description of the block internal structure, the type of composition and the topology of internal communications. For the sake of simplicity, and also to show how our approach can be applied to independently realized SysML models, we propose the SysML Distiller example described in [17]. The Distiller block definition diagram is depicted in Figures 3,4.
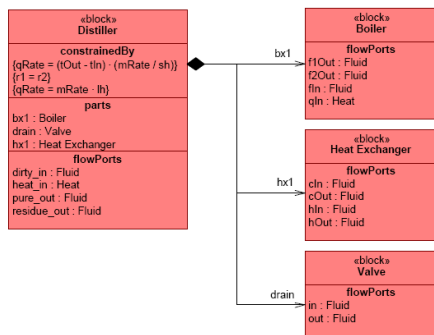
Figure 3.   : The Distiller Block Diagram [17]

A distiller is represented as a block composed of other blocks, including a Boiler, a Heat Exchanger and a drain Valve. A simple example of an internal block diagram (ibd) is shown in Figure 5. It shows how the data obtained as input by the distiller are processed and communicated by its component parts to obtain the output data on its output ports.
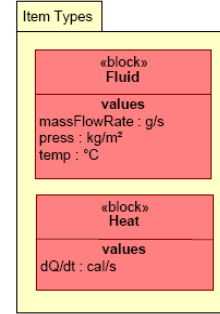
Figure 4.   : Port types specification [17]

Simulink [19], developed by MathWorks [20], is a commercial tool for the modeling, simulation and analysis of multidomain dynamic systems.
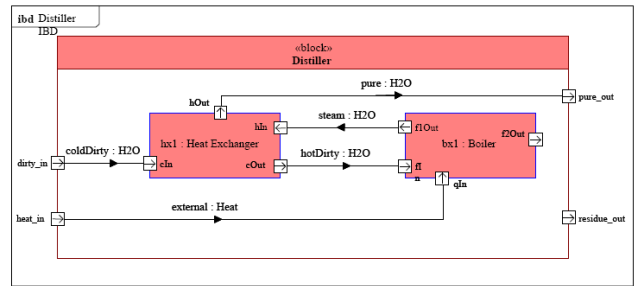
Figure 5.   : The Distiller Internal Block Diagram [17]

Simulink allows modeling and simulation of dynamic systems according to a synchronous reactive model of computation. A Simulink system is a network of blocks. Each block (in essence a Mealy machine) transforms an input function (of time) into an output function. The input function's domain can be a set of discrete points in time (discrete-time signal) or it can be a continuous time interval (continuous-time signal). Continuous blocks have a nominal sample time of zero, but in practice, they are implemented by a solver, executing at the base rate. Eventually, every block has a sample time, which is an integer fraction of the base rate. Simulink computes for each block, at each step, the set of outputs, as a function of the current inputs and the block state, and then, it updates the block state. A cyclic dependency among blocks where output values are instantaneously produced based on the inputs results in a fixed point problem and possibly inconsistency.

A fundamental part of the model semantics are the rules dictating the evaluation order of the blocks. Any block for which the output is directly dependent on its input (i.e., any block with direct feedthrough) cannot execute until the blocks driving its inputs execute. Executing a block means

computing the output function, followed by the state update function.

Of course, Simulink is supplement with a quite large library of systems and blocks that can be reused. In addition, simulink builds on top of the Matlab environment which offers an additional library of code implementations of mathematical, logic control and signal processing functions.

For these reasons Simulink is nowadays widely used in control theory and digital signal processing for multidomain simulation and Model-Based Design.

In Simulink, subsystems are a unit of encapsulation of behavior and the minimum unit for the code generation process. Simulink subsystems communicate through data ports, connected by signal links. Data ports and signals are typed by primitive types as defined by the tool (integers, reals, booleans) or composite types, defined by the user as Bus Objects.
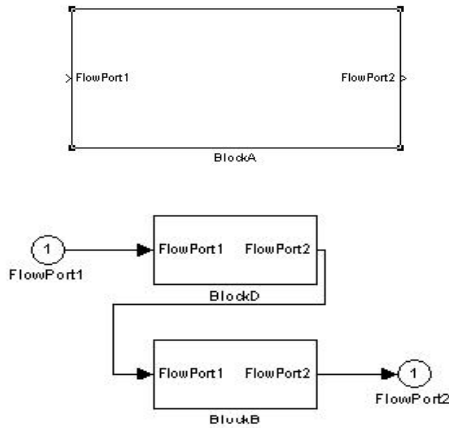

Figure 6. : An example of Simulink model

Figure 6-a depicts an example of a Simulink subsystem, called BlockA, having both an input Port (FlowPort1) and an output Port (FlowPort2). The internal view of the subsystem, showing how the data received from the input is processed by the subsystem blocks to produce the output signals is shown in Figure 6-b (in a way similar to a SysML internal block diagram).

IV. A SYSML TO SIMULINK TRANSFORMATION

Given that our objective is the transformation of a SysML model into a Simulink model (and vice-versa), the OMG QVT [12] language seems to be a suitable candidate. However, QVT assumes that both the source and target languages conform to meta-model specifications expressed by MOF. Unfortunately, while this holds for SysML, there is no available MOF description of the Simulink language. For this reason, QVT transformations are not applicable and we must therefore use a model-to-text transformation that translates a source SysML model into a Matlab model generation script. The output Matlab script can be processed by the Matlab engine to obtain a Simulink model having the same expressiveness of the source SysML model.

In the opposite direction, a matlab script parses the Simulink model and generates an XML model description that can be transformed into XMI (the standard input language for SysmML tools, supported by Topcased)

The SysML-to-Matlab transformation has been defined by means of the Acceleo template language [16]: an Eclipse implementation of the OMG MOF2Text Transformation Language [13]. For lack of space we do not provide a description of the Acceleo syntax and semantics which can be found in the related OMG standard.

However, we provide a short description of the rationale behind the transformation workflow together with a complete transposition of the transformation script.

According to the transformations taxonomy defined in [11], our SysML to Simulink transformation can be classified as endogenous, since the source and target languages conform to different meta-models, and horizontal, since both the source and target models reside at the same level of abstractions (no further details are added during the transformation.)

The following Table summarizes the mapping applied in our transformation workflow from SysML to Simulink.

SysML	Simulink	Note
Block	Block	Each SysML block is modeled with a Simulink Block
Blocks Composition	Subsystem Blocks	When a SysML Block consists of a number of other Blocks (its parts) it is modeled with a Simulink Subsystem Block
FlowPort	Input / Output Blocks	SysML FlowPort are modeled with Simulink Input and Output Blocks
FlowSpecification	Bus Object	SysML Flow Specification used to type FlowPort are modeled with Simulink Bus Objects

The transformation starts from the root element in the source model. First, it creates A Simulink Bus Object for each FlowSpecification associated with a FlowPort of the SysML source model. Then, the workflow continues by invoking the template generateRootBlock, aimed at generating the SimuLink representation of the SysML root Block in the model.

```
[template public generateModel(p : Model)]
  [comment @main /]
  [file (p.name.concat('.m'), false, 'UTF-8')]
  sys = '[p.name/]';
  new_system(sys) % Create the model
  open_system(sys) % Open the model
  load_system('eml_lib');
  libname = sprintf('eml_lib/Embedded\n
  MATLAB Function');
  portVocabulary={}
  portId={}
  blockStep=4;
  RootX=0;
  RootY=0;
```

```
    offset=150;
    [let classifiers:Set(Element)
     = p.allOwnedElements()]
        [for (cl:Classifier |classifiers)]
        [if cl.isFlowSpecification()]
            busCell = { ...
             {'[cl.name/]','Header
             File','Description', ...
             { ...
             {'a',1,'double',['['/]0.2
             0],'Real','Frame'}, ...
             {'b',1,'double',['['/]0.2
             0],'Real','Frame'}; ...
             }; ...
             }; ...
             };
            Simulink.Bus.cellToObject(busCell);
        [/if]
        [if isBlock(cl)]
        [if owners(cl)->isEmpty()]
        [generateRootBlock(cl)/]
        [/if]
        [/if]
        [/for]
        [/let]
        [/file]
[/template]
```

The template also triggers the generation of the root block ports (the generatePort template invocation), the parts composing the root block (the generatePart template).

```
[template public generateRootBlock(cl :
Classifier)]
    [if owned(cl)->isEmpty()]
    [else]
    [cl.name/]X=0;
    [cl.name/]Y=0;
    add_block('built-in/Subsystem',['['/]sys
       '/[cl.name/]' ['['/]/]);
    [generatePort(cl,'/'.concat(cl.name))/]
    [for (a:Association | owned(cl))]
    [cl.name/]X=[cl.name/]X+1;
    [cl.name/]Y=1+floor([cl.name/]X/blockStep);
    [generatePart(a.memberEnd-
     >at(2).type,'/'.concat(cl.name),
     cl.name.concat('X'),cl.name.concat('Y'))/]
    [/for]
    [generateConnector(cl,'/'.concat(cl.name))/]
    [/if]
[/template]
```

The generatePart template adds a new Simulink Subsystem Block to the block passed as parameter.

It also has the responsibility of generating its ports (generatePort template) its parts (by the recursive invocation of the generatePart template) and its connectors (generateConnector).

```
[template  public  generatePart(b:Type,  p:String,
x:String, y:String)]
    [let c:Classifier = b]
    [b.name/]X=0;
    [b.name/]Y=0;
    [if isBlock(c)]
    [if owned(c)->isEmpty()]
      add_block('built-in/Subsystem',
```

```
    ['['/]sys '[p.concat('/').concat(b.name)/]'
    ['1'/],'Position',['['/][x/]*offset
    [y/]*offset/2 [x/]*offset+100
    [y/]*offset/2+50 ['1'/]/]);
    [generatePort(c,p.concat('/').
      concat(c.name))/]
    [else]
    add_block('built-in/Subsystem',['['/]sys
    '[p.concat('/').concat(b.name)/]'
    ['1'/],'Position',['['/][x/]*offset
    [y/]*offset/2 [x/]*offset+100
    [y/]*offset/2+50 ['1'/]/]);
    [generatePort(c,p.concat('/').
     concat(c.name))/]
    [for (a:Association | owned(c))]
    [b.name/]X=[b.name/]X+1;
    [b.name/]Y=1+floor([b.name/]X/blockStep);
    [generatePart(a.memberEnd-
      >at(2).type,p.concat('/').
      concat(c.name),b.name.concat('X'),
      b.name.concat('Y'))/]
    [/for]
    [/if]
    [/if]
    [generateConnector(c,p.concat('/').
    concat(c.name))/]
    [/let]
[/template]
```

The generatePort template finally adds the specific blocks representing SimuLink in and out ports accordingly to the SysML source model FlowPort passed as parameter.

Moreover, when the FlowPort is typed by a FlowSpecification it generates the Matlab code that assigns it to the related Bus object.

```
[template public generatePort(b:Block,
name:String)]
    [for (port:FlowPort |b.ownedPort)]
    [let fd:FlowDirection  = port.direction]
    [if fd.toString()='in']
    add_block('built-in/Inport',['['/]sys
      '[name/]/[port.name/]'
      ['1'/],'MakeNameUnique', 'on','Position',
      ['['/]0 150 20 170['1'/]/]);
    [/if]
    [if fd.toString()='out']
      add_block('built-in/Outport',['['/]sys
      '[name/]/[port.name/]'
      ['1'/],'MakeNameUnique',
      'on','Position',
      ['['/]600 150 620 170['1'/]/]);
    [/if]
    portVocabulary{end+1}='[name/]/[port.name/]';
    portId{end+1} = get_param(strcat(sys,
     portVocabulary{end}),'Port');
    [/let]
    [if port.type.oclIsKindOf(FlowSpecification)]
    set_param(strcat(sys,'[name/]/[port.name/]')
     ,'UseBusObject','on');
    set_param(strcat(sys,'[name/]/[port.name/]')
     ,'BusObject','[port.type.name/]');
    [/if]
    [/for]
    [for (c:ConnectorProperty |b.ownedConnector)]
    [/for]
[/template]
```

V. A TRANSFORMATION EXAMPLE

We selected the SysML model of a Distiller proposed in [17] and depicted in Figures 3, 4 and 5 as an example for the application of our transformation scripts. We defined the Distiller model in TopCased by means of a SysML BlockDiagram and Internal Block Diagram (Figures 7 and 8). The only change to the original model is that FlowPorts are typed with FlowSpecifications instead of Blocks. This has been made to make the model more formal and unambiguous and facilitate parsing and interpretation.
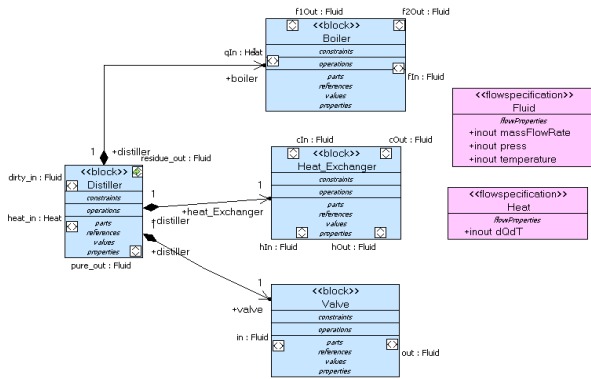
Figure 7.   : A Topcased Distiller Block Diagram
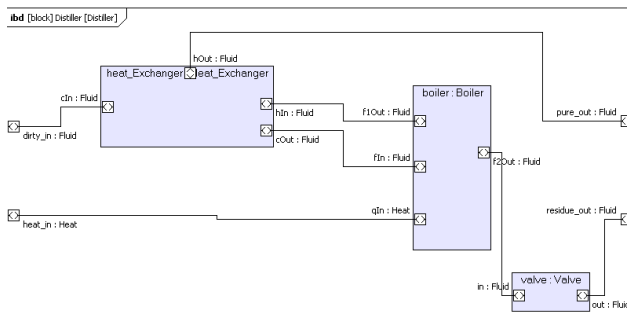
Figure 8.   The TopCased Distiller Internal Block Diagram

We applied the transformation workflow described in section IV to our Distiller SyML source model and obtained a Matlab script that generates a Simulink model conform to the source SysML model. For lack of space we do not show the resulting Matlab script (which can be derived from the Acceleo code and does not provide any additional information.). When executed in the Matlab environment, the Simulink model is automatically created as shown in Figure 9. The Distiller Simulink block is correctly created with its two input ports and two output ports. When the user double clicks on it, very similarly to what happens in Topcased for the same SysML model, another view opens (Figure 9) representing the internal model of the block. The Simulink internal block representation is very similar to the source SysML model (its internal block diagram) from which it has been generated. Each part is represented by a subsystem and subsystem ports are assigned a BusObject type according to the related SysML FlowSpecifications and connected as in the SysML model.
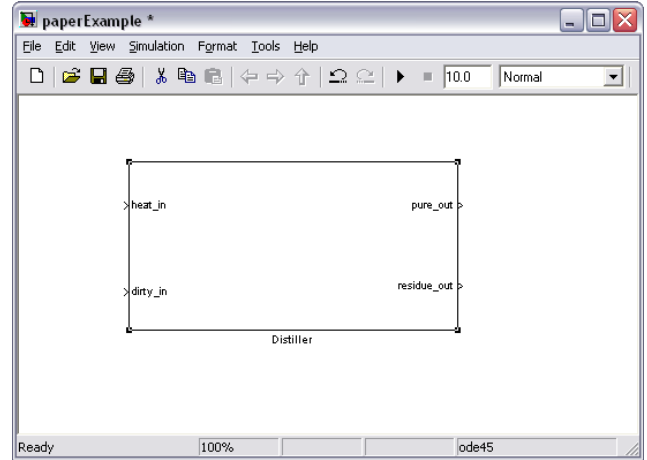
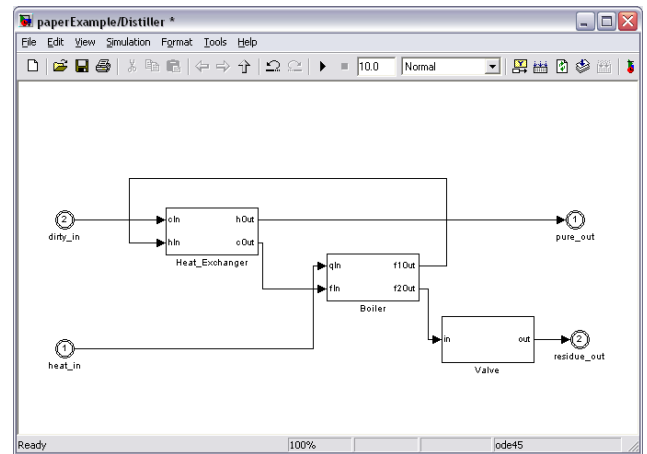Figure 9.   The Simulink Boiler Model

Figure 10. The Simulink Boiler Internal Model

VI. A MODEL DRIVEN ENGINEERING EXAMPLE OF ELECTRONIC DEFENCE SYSTEMS

The presented workflow has been designed and realized as part of a Model Driven Development process that Elettronica S.p.A [23] has adopted for designing its products. With the aim of providing a very high level demonstration of how this process can be applied to the design of complex systems in this section we provide a further example related to Electronic Warfare [21], the Elettronica S.p.A. core business, and particularly to the design of an ESM system [22], a passive sensor capable of detecting radars emissions.

An ESM system could be, simplicistically, modeled as being composed by the following macro components:

- A group of antennas: aimed at receiving radars signals which are possibly present in the surrounding electromagnetic environment;
- A receiver: capable of transforming analogic signals retrieved from the connected antennas into a stream of digital samples;

- A signal processing component: aimed at synthesizing from group of samples belonging to the same electromagnetic pulse a synthetic digital representation of each pulses called Pulse Descriptor Message (PSM);
- A data processing component: aimed at synthesizing from a group of PSMs emitted by the same Radar a synthetic representation of the Radar characteristics.
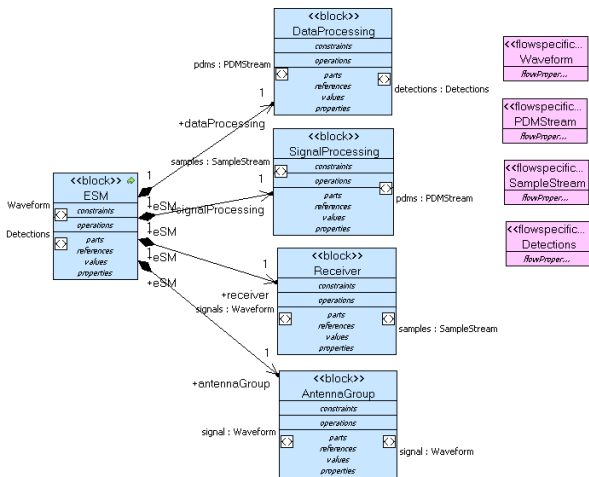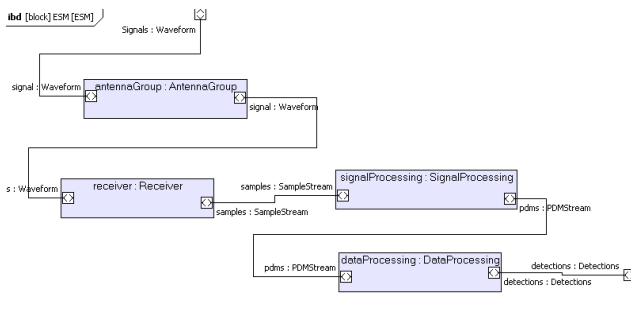

Figure 11. A SysML Block Diagram of an ESM System

Figure 12. A SysML Block Diagram of an ESM System

Figure 12 depicts The TopCased SysML model of a Block Diagram representing an ESM system. We designed the ESM Block as consisting of the following parts:

- An AntennaGroup Block, representing the ESM group of antennas. This block provides of two ports:
  - an input port representing signals illuminating the antennas
  - an output port representing the subset of signals intercepted by the antennas
- A Receiver Block, representing the ESM receiver. It provides two ports:
  - An input port representing signals retrieved from the AntennaGroup Block;
  - An output port representing samples produced by the Receiver Block

- A Signal Processing Block representing the related ESM signal processing component. It provides two ports:
  - An input port representing samples retrieved from the Receiver Block
  - An output port representing the stream of PSMs synthesized by the Signal Processing Block
- A Data Processing Block representing the related ESM Data Processing component. It provides two ports:
  - An input port representing PSMs retrieved from the Signal Processing Block
  - An outpot port representing synthetic detections created by the Data Processing Component

Figure 13 depicts the Internal Block Diagram of the ESM Block. This kind of representation is useful for System Engineers in order to design the system starting from its high level representation and then going deep within the inner architecture of its parts and sub components. Each of these parts can be further decomposed into sub-parts and more detailed by means of additional Internal Block Diagrams. Once these models are completed, System Engineers pass them along to HW, FW and SW Engineers involved with the realization of each system component. At this stage, Simulink models can be generated matching the original source SysML specification. HW, FW, and SW Engineers can proceed by working on the executable Simulink models performing simulation, test generation, test coverage analysis and code generation.
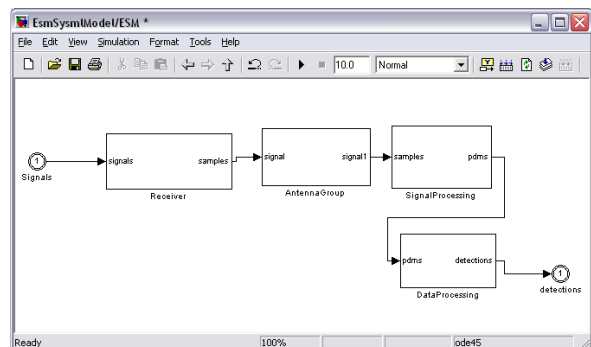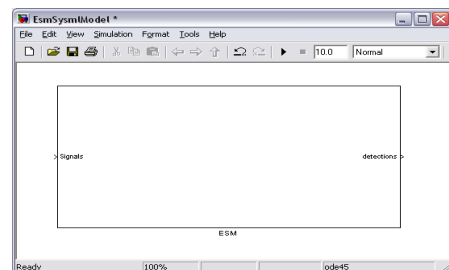

Figure 12: The generated Simulink Model automatically obtained from the related source SysML model

VII. Conclusions & Future Works

In this paper we have briefly described an automatic model transformation aimed at obtaining Simulink models conform to SysML source models.

This result represents just a first step in the definition of a unified modeling framework in which OMG languages (i.e. UML, SysML, etc.), particularly suitable to capture architectural aspects, are seamlessly integrated with Mathworks technologies (i.e. Matlab, Simulink, etc.) very adapt for the definition of executable models of signal and data processing.

Although the process we have defined has been implemented by means of OMG standards it is still slightly informal since the most formal way to transform SysML into Simulink should be a model to model, and not a model to text, transformation based on both the SysML and the Simulink meta-models. To this end in the future works we are going to provide ECORE defined meta-models for a subset of the Simulink languages enabling this kind of transformation too.

Our process also lacks of a transformation from Simulink to SysML that could be useful when Simulink models are already available and a SysML model is desired. We are already working on this kind of transformation that is basically based on matlab scripts capable of producing XMI description of SysML models conform to source Simulink models available in the Matlab workspace.

VIII. Acknowledgements

References

[1] D.C. Schmidt, "Model Driven Engineering", IEEE Computer V.39 n.2, p.25, February 2006

[2] Y. Vanderperren, W. Dehaene, "From UML/SysML to Matlab/Simulink: Current State and Future Perspective", Proceedings of the conference on Design, automation and test in Europe, Grenoble (France) 2006.

[3] Deutsche Bank F.I.T.T., "EDA Survey Results", 2005

[4] "ESL Now Survey", http://www.esl-now.com, 2005

[5] Electronics Weekly & Celoxica, "Survey of System Design Trends 2005", http://www.celoxica.com,2005.

[6] R. Soley, "Model-Driven Architecture", Object Management Group, 2000.

[7] The Object Management Group (OMG), http://www.omg.org/

[8] The MetaObject Facility (MOF): http://www.omg.org/mof/

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

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

[11] T. Mens, P. Van Gorp, "A Taxonomy of Model Transformation", Electronic Notes in Theoretical Computer Science, 152 (1-2), pp. 125-142

[12] Query/View/Transformation Language (QVT): http://www.omg.org/spec/QVT/1.0/

[13] The MOF To Text Transformation Language: http://www.omg.org/spec/MOFM2T/1.0/

[14] The Eclipse Modeling Project: http://www.eclipse.org/modeling/

[15] TopCased: http://www.topcased.org

[16] Acceleo: http://www.acceleo.org/pages/accueil/fr

[17] M. Hause, "The sysML Modeling Language", Fifth European Systems Engineering Conference (2006)

[18] Matlab: http://www.mathworks.com/products/matlab/

[19] Symulink: http://www.mathworks.com/products/simulink/

[20] MathWorks: http://www.mathworks.com

[21] S. A. Vakin, L. N. Shustov, R. H. Dunwell, "Fundamentals of Electronic Warfare", Artech Hous Radar Library, 2001

[22] Electronic Support Measures: http://en.wikipedia.org/wiki/Electronic_warfare_support_measures

[23] Elettronica S.p.A.: http://www.elt-roma.com

[24] G. Nicolescu, P. J. Mosterman, "Model Based Design for Embedded Systems (Computational Analysis, Synthesis and Design of Dynamic Systems), CRC Press, ISBN-10: 1420067842, November 24, 2009

[25] A. Benveniste, P. Caspi, S.A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The synchronous languages 12 years later. Proceedings of the IEEE, 91, January 2003.

[26] Model Driven Architecture: http://www.omg.org/mda/

[27] Kienhuis, B., Deprettere, E. F., van der Wolf, P., and Vissers, K. A. 2002. A methodology to design programmable embedded systems - the Y-chart approach. In Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation – SAMOS. Eds. Lecture Notes in Computer Science, vol. 2268. Springer, 18–37.

[28] A. Sangiovanni-Vincentelli, Quo Vadis, SLD? Reasoning About the Trends and Challenges of System Level Design, Proceedings of the IEEE, vol. 95, n. 3, pp. 467–506, March 2007.

[29] AUTOSAR. 2010. Autosar release 4.0 specifications. available at http://www.autosar.org/.

[30] Karsai G., Maroti M., Ledeczi A., Gray J., Sztipanovits J.: Composition and Cloning in Modeling and Meta-Modeling, IEEE Transactions on Control System Technology, 2003.

[31] F. Balarin, L. Lavagno, C. Passerone, and Y. Watanabe. Processes, interfaces and platforms. Embedded software modeling in Metropolis. In Proc. of the 2nd ACM EMSOFT, Grenoble, France, Oct. 2002.

[32] J. Martınez, P. Merino, A. Salmeron, "Applying MDE Methodologies to Design Communication Protocols for Distributed Systems", Proceedings of the First International Conference on Complex, Intelligent and Software Intensive Systems (CISIS'07)