Model-based development of embedded systems -
the MADES approach

Neil C. Audsley, Ian Gray*, Leandro Soares Indrusiak,
Dimitris Kolovos, Nikos Matragkas*, Richard Paige
University of York, York, U.K.

Abstract—This paper discusses the goals of the MADES project, which aims to use techniques of model-driven development to assist the development of complex embedded systems. Three main areas are covered by MADES - the development of embedded software; the development of embedded hardware; and the verification and validation of the system. This paper presents an early example of the MADES model transformations being used to generate synthesisable VHDL descriptions from high-level UML MARTE models.

Index Terms—Model Driven Engineering, Epsilon, MADES, Embedded Systems, Real-Time Systems.

I. Introduction

The architectures of embedded systems are becoming increasingly non-standard and application-specific. They frequently contain multiple heterogenous processing cores, non-uniform memory, complex interconnect or custom hardware elements such as DSP and SIMD cores. However, programming languages have traditionally assumed a single processor architecture with a uniform logical address space and have abstracted away from hardware implementation details. As a result, developing software for these architectures can be challenging. Equally, such systems are frequently deployed in high-integrity or safety-critical systems which require the highest levels of predictability and reliability.

The MADES Project is an EU-funded project that aims to use model-driven techniques to enable the development of the next generation of highly complex embedded systems, whilst reducing development costs and increasing reliability and predictability. In this paper we provide an overview of the MADES approach to model-driven development of embedded systems.

II. MADES Project Goals

The MADES project aims to develop the elements of a fully model-driven approach for the design, validation, simulation, and code generation of complex embedded systems to improve the current practice in the field. MADES differentiates itself from similar projects in that way that it covers all the phases of the development process: from system specification and design down to code generation, validation and deployment. Design activities exploit a dedicated language developed on top of the OMG standard MARTE (Modeling and Analysis of Real-time and Embedded systems) [4], and foster the reuse of components by annotating them with properties and constraints to aid selection and enforce overall consistency.

Validation activities comprise the verification of key properties of designed artifacts and of the transformations used throughout the development process, and also the closed-loop simulation of the entire system. Code generation addresses both conventional programming languages (e.g., C) and hardware description languages (e.g., VHDL), and uses the novel technique of Compile-Time Virtualisation to smooth the impact of the diverse elements of modern hardware architectures and cope with their increasing complexity.

All these aspects will be fully supported by prototype tools integrated in a single framework, and will be thoroughly validated on real-life case studies in the surveillance and avionic domains. The project also aims to develop a handbook to provide detailed guidelines on how to use MADES tools in the development of embedded systems and promote their adoption.

III. The MADES Approach

A conceptual model of the inter-relationships between the various artifacts of the MADES approach is illustrated in figure 1.

One of the main characteristics of the MADES approach is the use of model-driven transformations throughout the entire design process. These transformations are used to convert one or more input specifications into one or more output specifications. Model transformation languages are defined in a metamodel level and establish the relationship between source metamodel elements and target metamodel elements (see figure 2).

The MADES approach focuses on three areas:

- Generation of platform-specific embedded software from architecturally-neutral software specifications.
- Generation of hardware descriptions of the modelled target architecture.
- Verification of functional and non-functional properties.

Development effort starts with building design and analysis models using the MADES modelling language, which is based on a combined subset of OMG MARTE [4] (a UML profile for modelling real-time embedded systems).
The MADES approach. Such activities are present during:

- Specifications of functionality in an architecturally-neutral way.
- Descriptions of target hardware.
- Deployment diagrams that map functionality to hardware/software.
- Timing and non-functional properties for early and frequent verification.
- Code-reuse, component-based design, and maintainability.

Verification and simulation play a key role in the MADES approach. Such activities are present during:

- Verification of key properties on designed artifacts (for example, whether a system will meet a specified deadline, or be able to support a specified volume of data).
- Closed-loop simulation based on detailed models of the environment (for functional testing and early validation).
- Verification of designed transformations (from high-level system models down to low-level hardware/software implementations).

In order to provide verification and simulation in the MADES toolset, the Zot tool [5] is used. The verification phase aims to provide rapid and early verification of the system to reduce design time and guarantee correctness of the final system.

The MADES code generation phase allows the designer to model the target hardware at a high-level of abstraction and use deployment diagrams to map the input code (which is provided in an architecturally-neutral form) to elements of the target hardware. A technique called Compile-Time Virtualisation (CTV) [1], [2] is used to hide the complexity of embedded software development through the provision of a Virtual Platform (VP). The VP is an idealised view of the underlying hardware which provides a simple programming model that is compatible with the chosen source language. The result of this is that the programmer can write code as if it is to be executed on the VP, and CTV will then automatically retarget this code for execution on the actual platform. If the actual platform changes during development (for example, due to hardware redesign or changing requirements) the same input code can be automatically retargeted to the new hardware and does not manually ported.

Finally, the MADES approach also considers generation of synthesisable hardware descriptions from the hardware model. The deployment mappings of the code generation phase use a high-level description of the capabilities of the desired target architecture (for example, “three processors connected with a common bus, two banks of shared memory”). This can be reified into an unambiguous hardware description for implementation using the MADES approach.

The Epsilon platform [3] is used to implement both the model-to-model and model-to-text transformations used in the MADES approach. Epsilon (Extensible Platform of Integrated Languages for mOdel maNagement) is a platform for building consistent and inter-operable task-specific languages for model management tasks such as model transformation, code generation, model comparison, merging, refactoring and validation. Epsilon can provide traceability information produced by the various transformations, which is of paramount importance for embedded systems design due to the need to comply to particular standards such as the DO-178B Standard.

IV. Hardware generation example

This section presents the preliminary implementation of one of the three main areas of the MADES project, that of model-directed hardware generation. Following papers will discuss the areas of software development and verification.

The hardware generation approach is shown in figure 3. In this implementation, FPGAs are used to implement...
and test the generated architectures. The hardware architecture description is provided using a set of MADES hardware modelling languages. These languages are briefly described below and detailed in figure 4, later papers will formally define these languages:

- **H2**: A high-level model of the target architecture. Its constituent elements are processors, memory spaces, communication channels, clocks, and custom hardware elements. Architectural links (for example CPU → Memory) are used to connect elements. This model does not describe any lower-level details (such as bus topologies).

- **H1**: A refinement of H2. H1 is a lower-level model which codifies the bus topology of the system and specifies I/O. It still contains all information from H2, it is a true refinement. H1 does not define the specific types of each hardware instance. For example, at this level the model will still denote ‘processor’ rather than ‘Arm9’.

- **H0**: H0 instantiates the H1 model in terms of the MHS language (see below). Generated by model-to-model transformation from H1 using transformation rules and in-place refinement, known as ‘polishing rules’.

H0 demonstrates an equivalent level of abstraction to that of the Microprocessor Hardware Specification (MHS) language [9] used by Xilinx Corporation’s FPGA Development Tools. MHS files are generated by the MADES tools from an H0 model using model-to-text transformation. The Xilinx tool ‘platgen’ [8] is then used as an underlying HDL generator, as it can take an MHS description and generate a set of synthesizable VHDL files for implementation on an FPGA. This allows the generated architectures to be realised and tested. The development process is therefore:

1) Initial hardware model is specified in a high-level UML MARTE model.
2) This model is translated using the Epsilon toolchain with a model-to-model transformation into H2. The mappings for this are shown in table I.
3) H2 model is refined through H1 to an H0 model using the Epsilon framework.
4) The H0 model is converted to an MHS file using a model-to-text transformation.
5) The MHS file can then be passed to the Xilinx tools to generate VHDL.
6) The generated VHDL is then synthesised and implemented using the tools of the FPGA vendor.

The use of MHS files and Xilinx FPGAs is not required by the MADES toolchain, but it is a useful language to work with as it already has robust industrial-quality tool support available. If a different language or implementation fabric is required, another model-to-text transformation can be used. The modularity of the Epsilon framework assists the development of multiple output translations without affecting the high-level models.

Figure 5 shows an example result of the MARTE to H2 transformation. As can be seen, this translation maintains the same level of information as its classes correspond to MARTE’s hardware stereotypes (annotations and properties can also be carried over) and is used to convert the model into a form that can be used in the rest of the Epsilon-based MADES toolchain.

More translation work is required in the H2 to H0 transformation as H0 contains a greater amount of information than H2. This information is filled in using a template-based solution. For example, instances of H2’s processor class can be assigned a ‘type’ property that informs the translation process which specific processor type should be used to implement it. If this is set to a Microblaze soft processor [7] then the translation process will instantiate the Microblaze template. Templates are composed of MHS

<table>
<thead>
<tr>
<th>Model</th>
<th>Modelling level</th>
<th>Simulation level</th>
</tr>
</thead>
<tbody>
<tr>
<td>H2</td>
<td>Topology not modelled, totally-connected network assumed</td>
<td>Functional</td>
</tr>
<tr>
<td>H1</td>
<td>CPU, Mem, CANbus, DCache</td>
<td>Topology modelled, untimed, simple functional model reduces accuracy of simulation</td>
</tr>
<tr>
<td>H0</td>
<td>Arm9, DCache, DDR2</td>
<td>Complete hardware model, allows verification Cycle-accurate</td>
</tr>
</tbody>
</table>

**TABLE I**

MARTE to H2 mappings. A right arrow (→) denotes an association between two classes.
segments that describe the hardware element in question and any additional support peripherals that it requires. (The Microblaze example also instantiates an interrupt controller and clock generation logic.)

Templates are also used to specify bus topologies. The default template instantiates a single peripheral bus for each processor of the target architecture. If a peripheral is connected to multiple processor buses but only supports a single bus interface than a new shared peripheral bus is created and bus bridges used. The list of supported templates is currently small, but is being constantly expanded.

Processor address maps can be automatically determined (by successively assigning addresses from 0x00000000 upwards) or specified by the designer using properties in the MARTE model which are then carried into the M2 model by the transformation.

This preliminary implementation of the MADES hardware generation system is currently being used to assist development of the MADES modelling languages. Through experimentation it can be determined which hardware modelling features are important and if there are any features that are not already covered by MARTE.

V. Conclusion

The MADES project aims to use model-driven engineering techniques to aid the development of embedded systems. It uses a systems modelling language based on MARTE and SysML that allows the developer to express their system at a high-level of abstraction, and then to iteratively refine their design to reach the final implementation. MADES differentiates itself from similar work through three unique features. First, extensive use of model transformations is used to facilitate development and provide traceability. Second, verification and validation are key parts of the MADES design flow, allowing early and frequent verification of the system being developed. Third, Compile-Time Virtualisation (CTV) is used to assist the development of embedded software.

This paper has shown how the MADES model transformation framework is being used to generate synthesisable hardware descriptions of non-standard embedded systems. Whilst only a preliminary implementation, the translations can already produce complex hardware descriptions quickly from high-level system models with only minimal designer input. Model transformation techniques are also being applied in the MADES project to assist the mapping of software to the generated architecture.

References

Abstract—One of the major benefits of Model-Based Engineering (MBE) languages for Real-Time and Embedded (RTE) systems development is their capacity to allow the specification of complex non-functional properties and constraints in a declarative manner and the refinement of these properties towards the implementation. The UML profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) provides concepts to deal with non-functional aspects at different levels of abstraction as well as a concrete syntax for their textual representation: the Value Specification Language (VSL). The usage of VSL is quite straightforward for the specification of values. In the case of constraint specifications, using VSL may appear less intuitive, especially if the constraint concerns timing aspects. In order to let users benefit from the expressivity of VSL, this paper gives some methodological guidelines on the usage of VSL for the specification of time constraints. Propositions are illustrated via a simple component-oriented model example and put into action thanks to the UML modeler Papyrus and its plugin for MARTE.

Index Terms—MARTE, VSL, Time, Constraint

I. INTRODUCTION

Specification of values for properties of model elements is a frequent need in Model-Based Engineering (MBE). While this issue exists in general purpose modeling languages such as UML[3][4] (where OCL can provide a solution), it becomes crucial in the domain of Real-Time and Embedded (RTE) systems, where one key aspect of the modeled systems concerns the non-functional properties associated with model elements. For example, if a given model captures a hardware platform, associated hardware components must be characterized with non-functional aspects such as bandwidth, throughput or energy consumption (each characterization involving both the specification of a value and a unit) such that this information can be unambiguously used for further model-based design or analysis activities.

The Value Specification Language (VSL), standardized in the context of the UML profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE)[2], provides a formal solution to this concrete modeling issue. Coupled with the Non-Functional Properties (NFP) sub-profile of MARTE (which can be used to specify the complex types behind non-functional properties of a system), VSL covers all the expressivity needs implied by a typical RTE design flow.

Beyond specification of values (which is the most common use case), other usages of VSL are also possible. The specification of constraints is such an application area, where the specification of constraints can benefit from the expressivity of VSL for non-functional values. This use case, which is mentioned in the MARTE specification, simply relies on the fact that VSL is fundamentally a typed expression language. The intuitive idea is that a VSL expression denoting a constraint must be a Boolean expression.

While the principle is simple, putting it into action is not straightforward, especially when values manipulated in a constraint are related to timing aspects. The underlying complexity is mainly due to the way modeling constructs related to constraints and time are considered in the standard UML meta-model. As Time is one of the key considerations in RTE design, simple methodological guidelines are clearly required for users who may want to benefit from the expressivity of VSL for their application domain, while not being experts of the OMG standards.

The purpose of this article is to provide such methodological guidelines on the usage of VSL for specification of time constraints. These guidelines result from a careful analysis conducted in the context of the ITEA 2 project VERDE (http://www.itea-verde.org/). In section 2, we start by providing a brief overview of the VSL syntax and semantics and illustrate its usage on some typical non-functional property examples. Section 3 then provides precise modeling guidelines on the usage of VSL for specification of time constraints. Relationships with underlying UML concepts are highlighted and focus is given to the usage of the TimeObservation concept. In section 4, we illustrate the guidelines on the basis of a simple component-oriented model example, where a UML interaction (capturing message
exchanges between components and depicted in the form of a sequence diagram) is augmented with some time constraints specified with the help of VSL. Section 5 briefly positions VSL with respect to the Clock Constraint Specification Language (CCSL, also standardized in the context of MARTE), a language dedicated to the manipulation of time. Section 6 then concludes this article and sets objectives for future works.

II. AN OVERVIEW OF VSL

VSL proposes an expression language which might be used by any other UML-based specification interested in extending the base expression infrastructure provided by UML. This MARTE expression language is an extension to the “Value specification” and “DataType” concepts provided by UML. VSL deals with the following requirements:

- How to specify parameters/variables, constants, and expressions in textual form.
- How to specify composite values such as collection, interval, and tuple values.
- How to specify relationships between different parameters/variables, or constant values are defined.
- How different time values and assertions are defined in UML.
- How to specify arithmetic, logical, relational, and conditional expressions.
- How different time values and assertions are defined in UML.
- How to specify arithmetic, logical, relational, and conditional expressions.

VSL expressions can be used to specify non-functional values, parameters, operations, and dependency between different values in a UML model, as illustrated in Table 1.

<table>
<thead>
<tr>
<th>Value Spec.</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real Number</td>
<td>1.2E-3    //scientific notation</td>
</tr>
<tr>
<td>DateTime</td>
<td>#12/01/06 12:00:00#    //calendar date time</td>
</tr>
<tr>
<td>Collection</td>
<td>(1, 2, 88, 5, 2)    //sequence, bag, ordered set.</td>
</tr>
<tr>
<td>Tuple and choice</td>
<td>(value=2.0, unit= ms)    //duration tuple value</td>
</tr>
<tr>
<td>Interval</td>
<td>[1..251]      //upper opened interval between integers</td>
</tr>
<tr>
<td>Variable declaration &amp; Call</td>
<td>var1        //variable call expression.</td>
</tr>
<tr>
<td>Arithmetic Operation Call</td>
<td>+ (5.0, var1) //add operation on Real datatypes</td>
</tr>
<tr>
<td>Conditional Expression</td>
<td>((var1&lt;6.0)?(10^6):1) //if true return 10 exp 6, else 1</td>
</tr>
</tbody>
</table>

Table 1. Example of VSL expressions

UML modelers can also use VSL to specify non-functional constraints in their models. In RTE applications, some of these constraints deal with the timing aspect. For this purpose VSL provides a TimeExpression package that defines the semantics of timing concepts and an associated textual syntax specified by a BNF grammar. The TimeExpression package extends the UML to allow the capture of more sophisticated time expressions, constraints and observations. The next section proposes a modeling pattern putting these mechanisms into practice for expressing time constraints.

III. A MODELLING PATTERN FOR SPECIFYING TIME CONSTRAINTS WITH VSL

The following modeling pattern has been defined in the context of the ITEA 2 project VERDE, mentioned in the introduction to this article. This pattern consists in enriching a UML Interaction (which describes a set of observable communication exchanges between multiple communicating elements) by specifying time constraints related to communication events (graphically represented in a sequence diagram by the ends of messages). The expression of constraints rely on the usage of time observations (see UML Superstructure v.2.3 Section 13.3.30 TimeObservation), which are then further manipulated in a VSL expression actually constraining these time observations.

In the context of VERDE, this pattern has been modeled using the Eclipse Process Framework (EPF1), an eclipse-based tool supporting SPEM (the Software & Systems Process Engineering Metamodel OMG specification [5]). The following description therefore follows the decomposition implied by the EPF model. Section “Key considerations” identifies potential usages of stereotypes or concepts from the MARTE profile. Section “Input work products” indenitifies model elements considered as input of the modeling process. Section “Steps” identifies the various modeling steps implied by the modeling pattern. Section “Output work products” finally identifies the model element resulting from the modeling process.

A. Key considerations

This task may involve the following stereotypes or elements from the MARTE profile (see UML Profile for MARTE v1.1):

- <<TimedInstantObservation>> (see section 9.3.1.4), used to enrich UML time observations,
- <<TimedConstraint>> (see section 9.3.1.3), used to enrich UML constraints,
- VSL (see Annex B), used to capture formal expressions specifying the actual constraint body.

B. Input work products

- UML Interaction. A UML Interaction is a kind of UML behavior focusing on the description of exchanges between communicating entities. It is most of the time depicted as a sequence diagram.

C. Steps

- Create time observations: This step consists in creating time observations (in the context Interaction), so that they can further be referenced in VSL expressions, formally capturing timing constraints. Each time observation will typically refer to a communication event associated with a message from the interaction, or to an execution occurrence. Graphically, a time observation is represented by the symbol “@” followed by the name of the time observation. The fact that a time observation

1 http://www.eclipse.org/epf/
is actually bound to a communication event of a message will be graphically captured by having the time observation located at the corresponding end of the message.

- **Create a constraint**: In order to encapsulate an expression that will actually describe the timing constraint, a UML Constraint must be created. This constraint is typically owned by the context UML Interaction, and can additionally refer constrained elements (e.g., the time observations that will be manipulated in the VSL expression). Note that these additional references have no semantic impact on the VSL specification of the constraint, in the sense that it does not restrict the set of time observations that can be manipulated in the expression. Therefore, it can be seen as additional information making the model potentially easier to read or exploit.

- **Specify the constraint body with VSL**: Once the UML constraint has been created, a VSL expression can be encapsulated in it. Encapsulating the VSL expression involves the usage of an OpaqueExpression. The property language of the opaque expression must contain the string "VSL", and the property body must contain the VSL expression. Since properties language and body are ordered collections, the indexes of "VSL" and of the VSL expression (in their respective collection) must be the same (see Section 7.3.35 OpaqueExpression from the UML superstructure).

The VSL expression must be a Boolean expression (i.e., an expression whose evaluation will produce a result of type Boolean), which will typically make reference to time observations. For example, if the context interaction defines time observations @t1 and @t2, the following VSL expressions could be specified (this list is of course not exhaustive):

- o @t1 < @t2, which specifies that the event associated with the time observation @t2 must occur before the event associated with the time observation t1 (shortly, @t1 must occur before @t2)
- o @t2 > 11:43:45 2010/09/21, which specifies that @t2 must occur after a date literally specified
- o @t2 - @t1 < \{value = 15.0, unit = ms\}, which specifies that the duration between the occurrence of @t1 and occurrence of @t2 must be lower than 15.0 milliseconds
- o @t2 - @t1 < 15.0, which roughly specifies the same thing than the previous constraints, without specifying the time unit (i.e., it can be implicit from the context, or can be indirectly obtained from another model element, as illustrated in the last step of this task).

- **(Optional) Refine time observations with the MARTE stereotype <<TimedInstantObservation>>**: As explained in the previous steps, in the context of an Interaction, a TimeObservation is bound to a communication event (i.e., the emission or reception of a message), and can therefore be literally interpreted as a specification of the instant where a message is emitted or or received. However, this explanation only remains an interpretation.

In order to avoid any ambiguity on the interpretation of the event observed via the time observation, MARTE provides a stereotype : <<TimedInstantObservation>>. With the property obsKind : EventKind of this stereotype (possible values are start, finish, send, receive, consume), it is possible to indirectly characterize the event associated with the time observation. For example, if we have:

- o @t1 with <<TimedInstantObservation>> {obsKind = send}, a VSL expression such as @t1 > 17:25 that the emission of the event underlying t1 must be done after a literally specified date
- o @t1 with <<TimedInstantObservation>> {obsKind = consume}, a VSL expression such as @t1 < 17:25 that the event underlying t1 must be consumed before a literally specified date. This can be used to specify the validity date of a message.

- **(Optional) Refine constraints with the MARTE stereotype <<TimedConstraint>>**: From a given expression context, it is possible to determine if a constraint actually concerns a particular instant (e.g., @t2 > 11:43:45 2010/09/21) or a duration (e.g., @t2 - @t1 < \{value = 15.0, unit = ms\}). Determining if the constraint refers to an instant or a duration typically requires an interpretation phase (which can be automated since the VSL syntax is formally defined), with an inference mechanism exploiting the content of the expression (e.g., the time events it refers to and the operator which are manipulated) as well as the context in which it is specified.

MARTE provides a stereotype which enables to explicitly tag a constraint as an "instant" and/or a "duration" constraint: <<TimedConstraint>>, which extends the UML metaclass Constraint. By applying the stereotype on a Constraint, it is possible to specify how the constraint must be interpreted, using the property interpretation: TimeInterpretationKind (possible values are instant and duration). If interpretation is set to the enumeration literal instant, then the constraint is interpreted as a constraint on instant value. If interpretation is set to the enumeration literal duration, then the constraint is interpreted as a constraint on duration value.

Note that the stereotype <<TimedConstraint>> also inherits from stereotypes <<TimedElement>> (see Section 9.3.2.7 of the MARTE specification) and <<NfpConstraint>> (see Section 8.3.2.5 of the MARTE specification).

With the property on : Clock (inherited from TimedElement), it is possible to reference a clock, which can itself associated with a time unit (e.g., seconds, milliseconds, ticks, etc.). Considering a VSL expression such as @t2 - @t1 < 15.0, this can be used to indirectly specify the time unit behind the real literal "15.0".

With the property kind: ConstraintKind (inherited from NfpConstraint), it is possible to further characterize the timed constraint (typical values are required or offered). Required indicates that the constraint represents a minimum quantitative or qualitative level. Offered establishes that the constraint
represents the space of values that the constrained elements can afford.

D. Output work products
- UML Interaction with timing constraints

The next section illustrates the usage of this pattern in the context of a component-based architecture design. In addition, it is important to notice that the same modeling pattern can be applied using UML DurationObservation. In VSL expressions as well as in diagrams, duration observations are denoted using the symbol ‘&’ followed by the name of the observation. Note also that VSL allows the expression of instant or duration observations with an occurrence index. For example, we can express the i-th occurrence of the \( t_1 \) instant observation with the following syntax: \( @t_1(i) \). VSL also introduces an additional syntax for expressing jitters, i.e. a special duration expression that specifies an unwanted variation in the instants when periodic events should occur.

IV. SPECIFYING TIME CONSTRAINTS WITH VSL: EXAMPLE

This section illustrates the use of the MARTE VSL to specify timed constraints in a component-based architecture design. We consider a very simplified architecture of an on-board satellite software (OBSW).

The architecture consists of three high-level components, as represented in the composite structure diagram of Figure 1: one component for the ground station, and two components for the satellite software itself. The first component of the satellite software, swBus, deals with communications with the ground station (and possibly with other software satellite components that are not represented). The second component, swCore, implements the on-board software itself (altitude and orbit control, energy management, management of the on-board payload, etc.).

Communications between ground and satellite are done through telecommands (commands from the ground) and telemetries (data from the satellite) that conform to space communication standards. They can be considered as one way messages. The communications between the software bus and the software core are mostly operation invocations.

![Figure 1: Composite structure diagram of example](image)

Such software runs on real-time critical systems (as it is sent into space, where maintenance is very complex). Therefore, it is very important to be able to specify execution times for each operation in the workflow in order to correctly allocate the threads that will carry the execution of these operations.

Scenarios can be defined to describe communications between the different components, using sequence diagrams. Time constraints and VSL notations can be used in these diagrams to specify execution and communication times, as illustrated in Figure 3.

![Figure 3: Time Constraint Example](image)

In this scenario, the ground first emits a command. The instant when the command is emitted is identified by the time observation \( @t_{Emit} \). swBus propagates the command to swCore, by a synchronous invocation of method \( processCommand() \). The time observation \( @t_{EndOfProcessing} \) identifies the moment when the processing of the command by swCore is finished. The constraint illustrated in the bottom of Figure 3 specifies that the duration between the moment when the command is emitted by the ground station and the moment when the corresponding processing is performed must be lower than 20 milliseconds.

This expression first relies on the fact that in VSL, a time observation expression (such as \( @t_{Emit} \)) is of type DateTime, a primitive data type defined in the TimePackage of VSL representing a date. This data type defines operator ‘-’, whose return value is of type NFP_Duration. The result of \( @t_{EndOfProcessing} - @t_{Emit} \) can therefore be compared to \{unit = ms, value = 20.0\}, a literal specification of an NFP_Duration specified using the standard VSL syntax for tuples. The constraint finally relies on the usage of the MARTE stereotype TimedConstraint to specify that the constraint is related to a duration (property interpretation) and represents a required QoS (property kind).

Such specifications can then be used to refine the model of the system under design, for example by setting thread
configurations (periods, etc.) that match the architecture requirements with respect to scheduling analysis or execution simulations.

V. VSL VS. CCSL

Users interested in the description of more complex time relationships are encouraged to use CCSL[1] (Clock Constraint Specification Language). In CCSL, time is considered as a set of instances of time values making reference to Clocks. The language then provides mechanisms to describe relationships between instants or clocks. The VSL is actually one brick that must be combined with other elements (e.g. CCSL) to address the complete range of problematic of a realistic application architecture model.

We can use CCSL to enhance our architecture example. If we suppose that the satellite we modeled is not geostationary, then communications with earth can only occur when the satellite is above the ground station. This depends on the revolution period of the satellite. The Satellite has to transmit its data only when its position is vertical above the GroundStation (+/- 5 degree, as illustrated in Figure 4), otherwise the satellite is collecting data.

![Figure 4: Satellite communication frame](image)

Let us consider a state machine associated with the Comm component, which manages the state of the communication according to the satellite position. To model this in MARTE, we can use CCSL to specify a clock related to the position of the satellite with respect to the ground station (position angle). This clock would then be used to trigger the state machine and then activate communication when possible and data collection for the rest of the time. Considering this logical form of time typically goes beyond the scope of what can be specified with VSL regarding time.

VI. CONCLUSION

We have proposed a modeling pattern for expression of timing constraints using VSL. Relying on an EPF model of the pattern, we have precisely identified the various modeling steps involved as well as relationships with underlying UML concepts.

REFERENCES

[1] Time modeling in MARTE, Charles André, Frédéric Mallet, Robert de Simone, I3S, Aoste Project, I3S/INRIA.
Model Transformation-Based Verification of ArchMDE Software Architecture By Using UPPAAL
From TURTLE Profile to Blackboard Style

Nourchene Elleuch
MOSIC Research Team
University Of Sciences
Tunis, Tunisia
nourchene.elleuch@gnet.tn

Adel Khalfallah
MOSIC Research Team
University Of Sciences
Tunis, Tunisia
adel.khalfallah@fst.rnu.tn

Samir Ben Ahmed
MOSIC Research Team
University Of Sciences
Tunis, Tunisia
samir.benahmed@fst.rnu.tn

Abstract—ArchMDE is a formal software architecture-based approach for the development and the verification of software architecture of real time systems. It provides a way to obtain the structure and the behavior of software architecture with hard real-time constraints and in accordance with blackboard style. This paper presents an overview of this approach and focuses on the behavior transformation rules. The timed automaton in UPPAAL is chosen as the semantic unit that allows analysis and verification of both functional and timing requirements in early phases of the development lifecycle.

Keywords— MDE; Meta-model; Model Transformation, Architectural Style; Real Time System; UPPAAL; Timed Automata

I. INTRODUCTION

The development of Real Time Systems (RTS) requires high costs and long delays due to their inherent complexity. New development methods and tools that address the temporal and concurrent aspects of the RTS behavior and structure should be elaborated in order to improve RTS developers’ productivity. Software Architecture-centric Development (SAD) [14] and Model Driven Engineering (MDE) [7] seem to be good candidates to deal with the following resulting challenges: a) MDE involves the systematic use of models, meta-models and models transformations as essential artifacts to make the development process cost effective. For that, different viewpoints are defined to design models: Computational Independent Model (CIM), Platform Independent Models (PIM) and Platform Specific Models (PSM) and a set of transformations may then be performed to make the system more platform-specific; and b) SAD is centered on the idea of reducing complexity through abstraction and separation of concerns. The software architecture is considered as the backbone for any successful software-intensive system and it is recognized as a first-class element when modeling a system. As a result, we notice the emergence of architectural styles, such as blackboard and filter architectures that can be used to guide the modeling and the evolution of software architecture in order to solve the recurrent architectural issue. In addition, the use of architectural style continues to be a vehicle for the improvement of software quality.

RTS engineering is a domain in which MDE can be helpful, particularly in addressing the problems of platforms and environments evolution as well as the problems of systems properties validation. For instance, authors [4, 13] describe how MDE can be used for the code generation of RTS and the author [8] describes the possibility of RTS validation by using test models. However, RTS model driven development is at its infancy and many challenges are still have to be overcome. The expression of architectural design decisions is one of the challenges [12]. In fact, the definition of architectural styles and the way to enforce their use and validation are still unresolved problems. This paper describes our approach “Architecture-centric Model Driven Engineering (ArchMDE)” [5, 6] that we propose as a potential solution in response to these issues. In fact, ArchMDE is a component-based software approach that adopts MDE to define blackboard software architecture with hard real-time constraints. It assists the systematic construction of the software architecture based on the use of meta-modeling and model transformations processes. The blackboard architectural style is treated as a meta-model and the appliance of architecture design decisions is set in terms of model transformations. However, a meta-modeling process defines only the syntax and the static semantic of blackboard component model.

The specification and verification of the correctness of composed component behavior is still a difficult issue to resolve. We choose to endow ArchMDE by a formal foundation to verify the correctness of composed component behavior. Thus, the elements of functional
architecture defined by TURTLE profile [2] (class diagram for structure point view and activity diagram for behavior point view) are being equivalently transformed into their counterparts (blackboard component diagram for structure point view and timed automata in UPPAAL [3] for behavior point view). The resulting model can be verified and validated using UPPAAL model checker.

This paper is structured as follows. We present in section 2 a brief overview of ArchMDE approach. Since ArchMDE is based on meta-modeling and model transformations, we describe in section 3 the majority of ArchMDE meta-model. In section 4, we describe the way how the semantic differences, between activity diagram and timed automaton, are bridged in order to define a set of transformation rules between TURTLE and UPPAAL. We explain, in section 5, the transformation rules applied to generate a behavior architecture-specific model. We enumerate, in section 6, various properties that must be verified in the resulting model. We discuss related work in section 7 and we conclude by providing perspectives and areas for future improvements.

II. ARCHMDE OVERVIEW

As software systems become more complex, the overall system structure and particularly software architecture, becomes a central design problem. Hence, we share the view of the author [12], who highlights that the architecture should be considered as “a first class modeling citizen” and specified as platform-independent implementation. Therefore, architectural requirements must be dealt at the PIM level which is itself decomposed into two models: an Architectural-Style Independent Model (AIM) and an Architecture Specific Model (ASM).

AIM and ASM are system models that don’t have any information about technology-specific implementation. AIM exhibits a specified degree of architecture independence in order to be suitable with a number of different architectural styles. It also describes the functional architecture of RTS. However, ASM is a refinement of AIM in which technology-independent architectural considerations are introduced. The main purpose of the ASM is to make architectural styles explicit in the model. The distinction between the two models increases the reusability of the AIM and allows a more control over possible types of implementations.

The ArchMDE engineering process is based on a top-down approach that supports the analysis, the design of functional, software and hardware architecture, the implementation and the verification phases through six steps. The starting point of ArchMDE is the requirements model (CIM) from which the AIM is derived. The AIM defines the structure and the behavior of the RTS. With respect to architectural style, the AIM can be transformed into the Software Architectural-Style Specific Model (SASM) that takes into account both the functional, non-functional and architectural characteristics. Adding Hardware Architecture Specific Model (HASM) features will give Operational Architecture Specific Model (OASM). Finally, the Platform Specific Model (PSM) can be derived from SASM and/or HASM from which source code can be generated (Fig. 1).

III. SYSTEM MODELING AND META-MODELING IN ARCHMDE

ArchMDE takes into account both the structural and the behavioral aspects of architectural style. For that, we have compared, in [6], two languages Unified Modeling Language (UML) and Architecture Description Language (ADL) that are used to express the main concepts and connectivity rules of architectural styles. Instead of being faced to choose between ADL and UML languages, we wonder whether their merge can be considered to benefit
from their advantages and avoid their drawbacks. This is what our ArchMDE’s software architecture meta-models have considered (Fig 2).

On the one hand, we define a new meta-model that considers the syntax and the static semantic of architecture elements to design the software architecture structure. The blackboard architectural style is defined by meta-model, which enumerates the essential elements that can be used to create software architecture according to blackboard. ArchMDE generates a network of communicating components that interact with each other by using shared data objects, posted on blackboard component. “One writer and several possible readers” is the synchronization policy of communication model. The user component reads object values emitted during previous period of other user component. A detailed description of ArchMDE software architecture structure is available in [6].

On the other hand, the software architecture behavior modeling is also based on meta-modeling techniques and particularly on model transformation process. The TURTLE activity diagram defined in AIM layer is transformed into timed automata network that specifies the system behavior according to blackboard style. The correctness of software architecture behavior is ensured by using UPPAAL model checking. But, the most challenge of the transformation process is: how to bridge the semantic gaps between the two languages? For that, we provide the similarities and differences between TURTLE activity diagrams and UPPAAL timed automaton (Table 1) based on three behavioral characteristics: concurrency, time and reactive behavior.

Concurrency raises several challenges like scheduling, synchronization, and communication of tasks. In TURTLE, the synchronization is based on rendez-vous protocol with the possibility to data exchanged via gates and local declared variables (synchronization with or without communication). Scheduling protocol respects the composition operators defined into tclasses diagram. As an example, preemption operator is more prior than synchronous operator. As for UPPAAL, the synchronization between two (or several) processes (automata) is ensured by defining a global shared binary (or broadcast) channel. Over this channel, the data can be exchanged via globally declared shared variables (synchronization with or without communication).

However, the scheduling policy must be designed by another automaton to define the priorities of processes and to make sure that time critical processes with hard deadlines are not delayed. The time model in UPPAAL is a continuous time. Technically, it is implemented as regions and the states are thus symbolic, which means that at a state we do not have any concrete value for the time, but rather differences [1]. This is the same thing in TURTLE profile.

### TABLE I. SIMILARITIES AND DIFFERENCES BETWEEN TURTLE ACTIVITY DIAGRAMS AND UPPAAL

<table>
<thead>
<tr>
<th>Behavioral Characteristics</th>
<th>TURTLE</th>
<th>UPPAAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concurrency</td>
<td>binary synchronization by Rendez-vous read-write semantics</td>
<td>binary and/or broadcast synchronization with/without values passing</td>
</tr>
<tr>
<td>Time</td>
<td>Continuous time</td>
<td>Continuous time</td>
</tr>
<tr>
<td>Reactive Behavior</td>
<td>TURTLE Activity diagram with synchronization and temporal operators</td>
<td>Timed automata network with channel, global and local data/clock</td>
</tr>
</tbody>
</table>

### IV. TIMED AUTOMATON IN UPPAAL

#### A. Syntax and Semantic

The theory of timed automata is widely used to specify and verify the real-time systems. In this section, we briefly describe the syntax and the semantic of those automata used in the UPPAAL tool. An automaton is composed of a set of locations and transitions between these locations. A location can be tagged as urgent (time is not allowed to pass when the system in an urgent location) or committed (a committed state cannot delay and the next transition must involve an outgoing edge of at least one of the committed locations). It is also possible to define an invariant on location that specifies a possible conjunction of clock conditions. Furthermore, the edge between two locations can be labeled with synchronization operators, guards and/or assignment expressions. The later can imply the use of clock and/or integer variables.

#### B. Timed Automaton Metamodel

The meta-model of timed automaton in UPPAAL, as depicted in Figure 3, doesn’t describe the complete semantics of timed automaton. The main aim is to provide minimum concepts of timed automaton syntax, which can be used to describe a blackboard architecture behavior’s specification.

In UPPAAL, a system behavior is described by a set of processes that are considered as an instance of templates. The later is an extended timed automaton with parameters that can be of any type. A process can synchronize with others processes. This synchronization is often steered by defining binary, broadcast or urgent channel. The use of global variables allows the communication between processes. Moreover, each process can have its own local variables. The assignments to local or global variables can be attached to transition as so-called updates.
V. TRANSFORMATION FROM ARCHMDE TO UPPAAL

Each activity diagram defined in AIM layer is transformed into template in ASM layer. This transformation takes in account: the mapping between activity diagram elements and timed automaton and the characteristic of blackboard style that implies the introduction of blackboard and controller templates. We describe in the following sections the most important transformation rules.

A. Rule 1: ActivityDiagram2Template

Each activity diagram is transformed into template with the same name. Furthermore, we have defined, in the declarative part of template, a constant number that is used as identification.

B. Rule 2: Initial&FinalNode2Location

The initial and final nodes are transformed into two locations whose names are respectively id_s and id_f.

C. Rule 3: ActionNode2(Sub)Templates

In TURTLE activity diagram, the action node is used to design either internal or external synchronization on gate, variable assignment or method invocation. So, the transformation rules depend on the instructions for using the action node.

1) VariableAssignment2Edge&Location: An outgoing edge and a new location are created. The edge is labeled with the assignment expression. For that, we must declare the variable as local.

2) MethodInvocation2Edge&Location: Such as variable assignment, we define an outgoing edge and a new location. The method can be attached to edge as so-called updates. However, the architect must write the method algorithm in the declarative part of template.

3) ExternalSynchronization2Templates: If the action node is used to describe an external synchronization on gates g1/g2 in order to exchange data value (Fig. 4), then we must apply the following sub rules.

a) BlackboardTemplate: If the blackboard template (Fig. 5) is not created yet in previous transformation steps, then we must define it. The communication in blackboard architectural style passes through a shared blackboard template. However, this communication protocol implies some issues such as starvation, deadlock and mutual exclusion. For that, we use write lock that either allows multiple templates to access the blackboard objects in a read-only way, or it allows one, and only one, template at any given time to have write access to the blackboard object. When the user template needs to communicate with others, it must obtain a write lock that allows to put data in the blackboard object. After, the user template could add their data, by using two global variables obj[tid][j] and syncval[tid][j], and thus releases the lock. tid is the template identifier ranging 0 to NT-1 where NT is the total number of processes (instance of templates). j specifies the maximum number of objects that the template can be defined on the blackboard. To prevent a change request when a blackboard is empty, the auxiliary functions setBEmpty(), isEmpty() and ModifyBEmpty() are defined. All of them use the variable empty initialized at true. The variable is set (setBEmpty()) when an addObject is performed and unset (ModifyBEmpty()) when all objects data are read.

b) ControllerTemplate: The focus of the controller template (Fig. 6) is to handle the best user template for reading blackboard object (trigger channel and best()) function when a change occurs in blackboard process (change channel).
c) SynchronisationWithValueEmission2subTemplate: The write lock (WLock channel) is performed before the critical section is entered and the unlock (unlock channel) is performed after the critical section is left. By critical section, we mean any writing of data in the blackboard object. Only one process at any given time has a write access. We define a variable WriteLockedBy whose initial value is NT to check whether a process has write access rights. Infact, if the value of WriteLockedBy is different from the value of process identifier then the process must wait. The next transition checks the number of processes that gained a write lock. If the number is equal to zero then we assign the identifier value to WriteLockedBy variable. This operation is done by auxiliary function testWLock(Fig. 7). The write operation ends at LeaveWL.

![Figure 7. T1 Template](image)

Figure 7. T1 Template

d) SynchronisationWithValueReception2subTemplate: The controller template informs T2 template (Fig.8) that blackboard contains new data. Then, the T2 template is synchronized with blackboard via read channel to retrieve interesting data.

![Figure 8. T2 Template](image)

Figure 8. T2 Template

D. Rule 4: DeterministicDelay2SubTemplate

Fig. 9.a. depicts that something is interpreted after 5 time units. The corresponding UPPAAL template (Fig.9.b) is composed by two locations and transitions. The first transition updates a clock (c) to zero. Hereafter, we define an invariant which stipulates that a system is not allowed to stay in the same state more than 5 time units. The transition guarded by c>=5 has to be taken into consideration.

![Figure 9. Deterministic Delay with its transformation](image)

Figure 9. Deterministic Delay with its transformation

E. Rule 5: Non-DeterministicDelay2SubTemplate

Same as deterministic delay, the only difference lies in the transition guard (c>=0). This change allows us to model that something is interpreted at most after 5 time units (Fig. 10.b).

![Figure 10. Non-Deterministic Delay with its transformation](image)

Figure 10. Non-Deterministic Delay with its transformation

F. Rule 6: TimeLimitedOffer2SubTemplate

In TURTLE, this operator means that the action (g!x) is offered during a period which is less to 5. If the offer happens, the final node is interpreted. Otherwise, AD is interpreted (Fig.11.a). The first transition of corresponding UPPAAL template (Fig.11.b) updates a clock (c) to zero. Hereafter, the invariant (c<=5) stipulates that a system is not allowed to stay in the state more than 5 time units, so that only one of the two transitions can be taken. Either, the transition guarded by c=5 has to be taken and the subtemplate associated to AD must be interpreted or the T1 subtemplate defined in rule 3.d must be occurred.

![Figure 11. Time Limited Offer with its transformation](image)

Figure 11. Time Limited Offer with its transformation

VI. Validation of Transformation Model

The presented model transformation method has been practically validated through a Cruise Control System case study. At the AIM layer, the system is designed with TURTLE profile. Then, it is transformed into a component diagram for a structural point of view and a network of timed automata from behavioral perspective. To validate the transformation rules, we classify properties into two categories. The first category deals with properties that are specific to blackboard style like deadlock and starvation free. The second category treats the properties which are specific to application domain like safety and liveness requirements.

We define a priority for channel to prevent a starvation problem. The Write Lock and read channels have the same priority, but the change channel has a higher priority. This results in the possibility of writer starvation. For that, the change channel is guarded by a race condition which implies that the transition being fired only when the condition becomes true. This condition requires the invocation of reader in case the buffer is either full or not empty. The absence of deadlock is checked by a single query (A[ not deadlock]). As an example, we describe one bounded liveness requirement that the cruise control has to respect: when the brake pedal is pressed; the cruise control should be turned OFF within a period of time not exceeding 50 ms. This requirement is checked by a...
query {obj[writeLockedBy][0]==PedPress} → Reg.OFF and Reg.h<=50). The result is OK.

VII. RELATED WORK

Many different methodologies for the platform-independent design and the development of RTS have been proposed in the literature. For that, the adoption of executable UML has been widely studied. For example, xtUML [10] defines an executable and translatable UML subset for embedded RTS, allowing the simulation of UML models and the C code generation. However, there is no support to formal verification tools in xtUML. Similarly, the Accord|UML methodology [8] uses executable modeling which suggests that the validation process is based on simulation. HIDOORS [15] encompasses the possibility to automatically generate Java code from a PIM and supports the analysis and validation of real-time properties by using WCET analysis. The approach of SAE, based on the emerging standard AADL (Avionics Architecture Description Language) [9], provides a mean to specify both the software and hardware architectures, to map software into hardware elements (operational architecture) and to produce component implementations. Also, this approach lacks formal support to define behavior model. The MARTE profile doesn’t introduce or target new specific analysis techniques. Rather, it aims at representing well-known models from schedulability and performance analysis theories into a UML framework, based on transformation tools [16]. However, to our best knowledge, no experiences of architectural styles modeling using these approaches are currently published yet.

VIII. CONCLUSION

In this work we have presented ArchMDE approach, which focuses on the RTS development and verification. The combination of architecture-centred, model-driven paradigms and formal approaches are employed as an RTS development process and constitutes the contribution. We have proposed a way to transform the TURTLE elements into blackboard architecture elements in order to address the challenge of designing and adapting software architecture for RTS. The introduction of the architectural viewpoint has several advantages, which mainly are:

- The architectural style for RTS provides means for representing components and leads to a rigorous and clear software architecture.
- The structure and the behavior of software architecture are automatically generated by executing a mapping process.
- The software architecture allows for design decisions expression and enables the comprehension of the system at a higher level of abstraction.
- The formal behavior of software architecture increases the quality of RTS by providing a way to check logical and time consistency.

REFERENCES

Abstract

This short paper presents the design, implementation, and initial results of a tool that extracts from UML models the schedulability analysis data that are necessary for the application of the MAST set of tools on them. The input analysis models represent concurrent, distributed real-time systems that are formalized in UML annotated with the extensions proposed by the UML Profile for MARTE; the OMG standard profile for the modelling and analysis of real-time and embedded systems. The experiences performed with the modelling elements that have been selected to support the underlying analysis methodology have led to the rising of concrete issues, which have been sent to the OMG for the improvement of the standard, and are also briefly described here.

1. Introduction

Model-based software development is one of the most promising software engineering approaches, since using reusable, configurable, and composable models may help significantly in the separation of concerns, increasing the efficiency, but also the quality of software.

In the case of applications with real-time requirements, a model-based methodology can help by simplifying the process of building their temporal behaviour analysis models. These models constitute the basis of the real-time design and the schedulability analysis validation processes. With that purpose, the designer of a real-time system must generate, in synchrony with the models used to generate the system’s code, an additional parameterizable model, suitable for its timing/schedulability validation. In the approach that we assume here, these analysis models are to be derived from the high level design models annotated with a minimum set of real-time features taken from the requirements of the application in which they are to be used. In analogy to the generation of the application’s code, the analyst, or application designer, have also to define the set of real-time sub-models, and build the complete real-time analysis model of the application.

The objective of this paper is the identification of a practical set of constructs in the Schedulability Analysis Model of MARTE in order to construct and extract the input model necessary to perform schedulability analysis with the MAST set of tools [2].

A discussion of the process followed for the modeling and the organization of the timing characteristics of a distributed, concurrent, reactive real-time system in a UML based methodology may by found in [1]. This short paper concentrates on (Section 3.) the modeling elements equivalencies for the generation of the output MAST analysis models, (Section 4.) the description of the technologies used for it, (Section 5.) the issues found in MARTE for the representation of such systems, and finally some conclusions and future work.

2. Related work

Some other tools have been realized in this direction. The closest in style and modeling capabilities is the RSA plugin to perform schedulability analysis by means of RapidRMA [5]. The version of the tool that is available has some limitations: it supports only scheduling analysis for mono-processors, with periodic and sporadic events (through sporadic servers). It does not provide support for multi-processors and distributed systems. RapidRMA and IBM RSA are not integrated thought the GUI. Moreover, there is no automatic launch of RapidRMA after the input files are generated. It requires a manual operation. The current implementation does not offer any feedback capabilities from RapidRMA to the UML modeling tool. All the analysis results can be exploited within the tool only. Similar limitations plus a lack in modeling guidance is provided by the tool [6] for representing Cheddar models with MARTE. The explanatory document shows how MARTE concepts can be matched to those used in Cheddar in order to do analysis on models and proposes model transformation solutions using ATL.
Other approach for timing verification is the use of specific clocks and timing annotations in MARTE and analyse them using Time Square [7]. Timing verification is pursued by using the equivalence of the behavioural models to a set of instants analyzed in a time-driven scheduled approach. In [4], Marcos et al. deal with modeling for verification using automated test generation and simulation tools.

3. Analysis models

The UML Profile for MARTE [3] brings a large number of modeling constructs and concepts that may be used for realizing schedulability analysis in a variety of ways. The tool here described uses a subset of those modeling constructs which is sufficient and adequate for enabling early V&V and the iterative use of the analysis models created.

In order to cope with complexity, to manage the risks associated to the research and the development of tools efforts, and also to make better use of the modeling resources offered by MARTE, we assume that the complete design/specification/analysis problem is divided in two challenging but achievable steps. One comprises the definition and manipulation of what we will denominate the "analysis models". The other one is the specification and automation of implementation oriented "design models".

This paper deals with the tool that exploits the analysis capabilities in MARTE: essentially the SAM, GQAM, and GRM chapters. In a model-driven approach, these analysis models are meant to be generated in a (semi or fully) automated way by the use of model transformations, which take the elements in the High Level Application Modeling chapter: RtUnit and PpUnit combined with the real-time features there described.

The first and more relevant problem has been the definition/selection of which elements in MARTE are to be used in the creation of the schedulability analysis models. These elements are the basis for the tool that has been developed for the generation of MAST analysis models taken from UML+MARTE annotated analysis models. Following previous research efforts [1], MARTE provides concepts to structure the analysis models using three main categories: The platform resources (a), the elements describing the logical behaviour of the system constituent parts (b), and finally the real-time situations to be analysed (c). The platform elements are modelled as a set of structural elements with stereotypes annotated on them. Figure 1 shows an example of the usage of these elements in the modeling of a tele-operated robot distributed platform. In a similar way, the logical components are modelled by means of regular
classes and operations stereotyped as SaStep. Finally, the Real-time situation is modelled as a SaAnalysisContext. As shown in Figure 2, the end-to-end flows that described scenarios are modelled using activity diagrams.

Though the precise mapping from MARTE to MAST elements is described in the documentation accompanying the tool, here we summarize a condensed view of the MARTE elements proposed for their use in each of these three main categories.

### 4. Technologies used

This effort has been realized using the technologies provided by PapyrusUML as graphical tool, the UML2 plugin as model repository, and the Acceleo plugin for the extraction of text from the UML2 models plus a significant number of Java functions. For the recuperation of results the classical XML to Java technologies have been used. The code used as well as the scripts created are shared as open source. An initial version with support for activity diagrams and composition of independently characterized timed behaviours is available from http://mast.unican.es/uml/marte2mast.

### 5. Enhancements to MARTE

The issues that are in their way to be sent to the OMG are basically related to the support for hierarchical scheduling (GRM) and the distinction between the typical context switch and the one used for servicing interrupt routines. (SAM)

From the tooling perspective the modelling element that has been used for the specific purposes of the invocation of the tool is the ContextParams attribute of the SaAnalysisContext stereotype.

This field has multiplicity * and is hold in an NFP_String type. The name it has in the Papyrus implementation of the profile is “context”, which is smaller and will be suggested for enhancement. The concrete parameters defined in it are:

- “invoke”: (true or false) defines whether the automatic invocation of the MAST tool shall be done or not.

### Table 1

<table>
<thead>
<tr>
<th>Platform Resources</th>
<th>Behavioral Models</th>
<th>Real-Time Situations</th>
</tr>
</thead>
<tbody>
<tr>
<td>GaResourcesPlatform SaExecHost * SaCommHost * SaSharedResource * SchedulableResource *</td>
<td>GaWorkloadBehavior GaScenario SaStep * SaCommStep</td>
<td>SaAnalysisContext * GaWorkloadEvent * Allocate Allocated Assign SaEndToEndFlow * SaSchedObs GaLatencyObs *</td>
</tr>
</tbody>
</table>

* Elements used in the extraction tool in current version.
- recoverResults: (true or false) defines whether the results obtained from MAST in its results file should be inserted back into the UML model or not.
- overwriteResults: (true or false) defines whether the response times annotated in the Steps should be replaced by the new ones or appended with the “mode” of the recently executed conversion for analysis.
- overwriteOutputModel: (true or false) indicates whether the UML output model should be overwritten or a new one need to be created to hold the results from the previous analysis.
- modeID: (string) indicates the name that wants to be used for the “mode” attribute that will identify the values resulting from the analysis to be performed after the execution of the tool. This id is going to be used in the name of the resulting UML model file also. By default a combination of date and time of the tool execution is used as the mode of the analysis context.

These parameters will be discussed in the OMG forum for its inclusion in the next versions of the standard.

6. Conclusions and future work.

Considering the prospects of the OMG’s UML Profile for MARTE as a modelling standard for analysis tools interoperability, it seems reasonable to look for model based strategies that link it with modeling intensive activities. And a clear semantics for the High level application modelling is the basis for automating the process of having timing analysis results quickly in the development life cycle. The issues that are in their way to be sent to the OMG are basically related to the support for hierarchical scheduling (GRM) and the distinction between the typical context switch and the one used for servicing interrupt routines (SAM).

The extraction of MAST analysis models from the UML+MARTE schedulability analysis specific models is a first demonstrable step in the direction pointed out by this effort. The tool is shared as open source with GPL license: http://mast.unican.es/umlmast/marte2mast

From the real-time and embedded systems research community perspective, this effort constitutes a step to get the effective exploitation of the capabilities of the available analysis and verification techniques, which despite the efforts in dissemination, have not yet reached an audience large enough to reward the many years of work in the field.

The next effort that need to be addressed is the required fixing in the semantics and models of computation of the RtUnits and PpUnits and the definition of the transformations in order to automate the construction of the schedulability analysis models from the restricted design models.

References

Safe Design of Dynamically Reconfigurable Embedded Systems

Xin An*, Abdoulaye Gamatié**, Éric Rutten*
* INRIA Rhône-Alpes, Grenoble, France ** LIFL/CNRS - INRIA Lille Nord Europe, Lille, France

Abstract—Dynamically reconfigurable embedded systems are more and more attractive with the high need to adapt embedded systems regarding frequent environment changes, better execution performances and lower energy consumption. This paper presents an approach for the safe design of these systems. The UML standard MARTE profile is adopted for the design. The resulting models are transformed into formal models (e.g. synchronous programs), by means of which, two levels of analysis are carried out: the system configuration analysis concerning their functional and nonfunctional properties by applying an abstract clock analysis, and the synthesis of a correct controller for system reconfiguration by using discrete controller synthesis to enforce the desired behaviors, and decide and trigger reconfigurations. At last a case study is provided to illustrate our approach. This study is achieved in a co-design framework, referred to as GASPAR2.

I. INTRODUCTION

Over the recent decades, there has been an increasing requirement for embedded systems to be able to dynamically adapt to their environment variations. Typically, a surveillance embedded system for street observation must adapt its image analysis algorithms according to the luminosity of the weather. The need for adaptivity may also come from the execution platform to provide a better performance or to reduce energy consumption in a system. For instance, for data-intensive algorithms such as the discrete cosine transform, a hardware accelerator gives a more powerful execution in terms of performance than a processor. Further motivations for adaptivity are the need for coping with different protocols and data-coding standards, e.g., in multimedia embedded applications.

Meanwhile, with more and more new features integrated into embedded systems, their design complexity has escalated. For example, in smart phones, multiple applications are made available for users. This leads to a real challenge about cost-effective and safe design methodologies of dynamically reconfigurable embedded systems. Firstly, design correctness issues must be addressed to ensure system reliability in every possible configuration. Secondly, reconfiguration correctness must also be established to safely control the variation between system configurations.

A. Contribution of the paper

We present a methodology for the safe design of dynamically reconfigurable embedded systems. Safety in the design is approached by using formal methods to verify the design, and to generate (part of) the controller triggering reconfigurations. It goes according to the following steps:

- **Step 1: system design using the MARTE profile.** The considered systems are modeled with the UML standard profile MARTE [8] that provides a rich set of concepts for the design of embedded and real-time systems. Both software application and hardware execution platforms are represented with adequate concepts. Then, different software/hardware allocations can be described. For all these aspects of a system, various configurations can be specified and modelled.
- **Step 2: system analysis for correct and efficient execution.** The system models resulting from the previous step are now analyzed from two main points of view: analysis of each system configuration and synthesis of a reconfiguration controller that enforces the designer-specified system properties. These analyses are achieved based on a transformation of MARTE models towards formal models. Here, we consider the formal tools and techniques provided by the synchronous technology [2].

- **Step 3: composition of initial, uncontrolled design with synthesized controller.** The results obtained from system analysis are now integrated to initial system models in order to define a correct design with respect to system requirements. This is done by composition of the original uncontrolled system and the synthesized controller.

The above methodology is currently under construction and is expected to be set up within a co-design framework, GASPAR2 [5], defined for high-performance embedded systems. A complementary approach to this work concerning the implementation level integration of the generated controller in specific FPGA platforms is proposed in [9].

B. Outline

This paper is organized as follows: Section 11 presents the design concepts regarding modeling techniques we have adopted for our methodology. Section 111 describes the analysis
techniques concerning system configuration and reconfiguration aspects respectively. A case study is presented in Section IV to illustrate our proposal. Finally, we conclude our paper in Section V.

II. DESIGN CONCEPTS
A. Modeling of reconfigurable systems with MARTE

We use the UML standard profile for Modeling and Analysis of Real-Time Embedded systems (MARTE) [8] to model embedded systems. MARTE offers a rich set of concepts to describe different features of reconfigurable embedded systems. The General Component Modeling (GCM) package is used to define general aspects such as algorithms in the application software part of a system. The Hardware Resource Modeling (HRM) package is used to describe hardware architecture, e.g. processors and memories. The Allocation package serves to define software/hardware mapping. Furthermore, for data-intensive applications such as image or video processing, data-parallel algorithms and multiprocessor execution platforms are described with the Repetitive Structure Modeling (RSM) package. All these packages are useful in the description of each system configuration. Concerning reconfiguration modeling, we also need additional features: Configurations, which is used to describe different scenarios, or modes, of a system, and UML Finite State Machines, which is used to describe configuration switches. All these concepts are available within the Papyrus tool used for modeling in GASPAR2 environment.

B. Modeling each configuration with a synchronous language

The synchronous approach [2] has been proposed in 80s to provide a rigorous mathematical semantics for the safe design of embedded real-time systems. The synchrony hypothesis means that there is a notion of logical instant, like a cycle or a step, within which computation and communication are made. Since then, synchronous languages, e.g. ESTEREL, LUSTRE and SIGNAL have been developed for and widely applied in the area of embedded systems. In our approach, we mainly consider SIGNAL [4], which adopts a multi-clocked philosophy for modeling: the system behaviors are described using relations between the values of observed events, and the occurrences, also referred to as abstract clocks, of these events.

A modeling approach from MARTE models to synchronous equational models has already been proposed in [6] in GASPAR2. The resulting models take into account platform and environment constraints on embedded data-intensive applications specified in MARTE. They are therefore translated towards different synchronous dataflow languages such as LUSTRE and SIGNAL.

C. Modeling reconfigurations with the BZR language

BZR [10] is a mixed imperative and declarative programming language, which expresses system behavior in terms of automata (imperative part) and specifies given properties to be enforced by using contracts (declarative part). The compilation of BZR automatically synthesizes a controller enforcing safety properties. This controller is then re-injected automatically into the initial BZR program so that an executable program can be generated (in C or Java) for execution. The automata and contracts in BZR can be generated from the state machine and mode switch components specified in MARTE.

III. DESIGN ANALYSIS TECHNIQUES
A. System configurations

The validation of system configurations is addressed by using the synchronous models. Properties of interest include functional properties that are addressed with the SIGNAL compiler: syntax and type analysis, data-dependency analysis, abstract clock analysis, and automatic code generation, exploitable in standard simulation environments such as GtkWave. Among these facilities, abstract clocks are very useful for characterizing components interaction within a configuration. Such an interaction quite depends on environment constraints and software/hardware allocation choices. Then, the compiler analyzes abstract clock constraints to check implementability criteria. This is commonly known as clock synchronizability analysis. Further important analysis techniques are available in SIGNAL, such as the temporal performance evaluation achieved by co-simulating a program P and its associated temporal interpretation T(P), i.e. another program obtained automatically from P. The program T(P) computes an approximate execution time of P according to a given system configuration.

B. Enforcing system reconfiguration correctness

Instead of addressing the correctness of reconfiguration by exploring control automata with model-checking, we rather adopt a more constructive method by enforcing reconfiguration correctness with discrete controller synthesis (DCS).

DCS is a constructive and automated method ensuring required properties on a system. It exploits transition system models, and is originally defined in supervisory control of discrete event systems. Inputs are partitioned into uncontrollable and controllable ones. The uncontrollable inputs typically come from the system’s environment, while the values of the controllable inputs are given by the synthesized controller. It also requires a specification of a control objective which needs to be enforced by control. DCS produces a controller which gives the constraints on controllable events, w.r.t. the current state and the uncontrollable inputs, such that the resulting controlled system satisfies the given objective. The synthesized controller is maximally permissive, and the most possible behaviors are kept. DCS has been defined and implemented as a tool integrated with the synchronous technology: the SIGALI tool [12] and BZR.

IV. CASE STUDY: CM PLAYER
A. Informal description

We consider a simple continuous multimedia (CM) player system (modeled in Figure 1 extracted from [14]). The CM server (or the synchronization component) takes as input the
streams of video and audio (CM) data packets from corresponding CM sources, synchronizes and assembles data from several packets into synchronized playable units, calculates the system time at which frames should be played, and dispatches them to output devices (e.g. the screener to play video and the speaker to play audio). The video stream is composed of a sequence of JPEG frames whereas the audio stream is captured in the form of Sparc audio.

**Figure 1.** The system configuration: LipSync

Since different temporal relations between media objects can be defined according to application specifications, in this paper, we suppose that the synchronization component has two modes or algorithms dealing with two different temporal relation specifications (taken and adjusted from [3]) respectively as shown in Figures 2 and 3. Both of these specifications are specified by using the reference point synchronization model [3]. Each block of the media is a logical data unit (e.g. frames for digital video) defined by the designer. Specially, a LDU is seen as a logical processing unit for media sources.

**Figure 2.** The lip synchronization specification

As we can see form Figure 2 (resp. 3), the lip synchronization algorithm (resp. slide show synchronization algorithm) each time takes an audio unit and two video units (resp. one slide and four audio units) for processing and assembling a single playable unit.

**B. Step 1: Design of the CM Player**

1) **Modeling of system functionality:** With two different execution modes for the synchronization component, we have two functional configurations as modeled using MARTE in Figure [1] and [4] namely LipSync and SlideShowSync. The stereotype `<<configuration>>` is used to represent a specific configuration with the name labeled below, which is associated with the value of the `mode` noted on the top right to indicate when the configuration is active.

2) **Modeling of system hardware platform:** For the hardware implementation of this simple CM player system, we consider a hardware architecture composed of a processor, a hardware accelerator and memory devices as modeled in [13] (see Figure 5). The CM sources are realized through sensors (e.g. a camera for video data and a microphone for audio data) which have dedicated media conversion units to produce the required data format for further processing. And we assume that each sensor has two different processing frequencies (15 and 45 MHz). A processor and a hardware accelerator, with frequency values 30 and 45 MHz respectively, are provided for the media synchronization processing. And the screener and speaker are realized as actuators, provided with two processing frequencies (15 and 45 MHz) as well.

**Figure 4.** The system configuration: SlideShowSync

**Figure 5.** The hardware platform model

The Hardware Resource Modeling package of MARTE is used here to describe the architecture. Figure 5 gives the details of our modeling.

3) **Modeling of mapping:** The mapping of the CM system onto the hardware execution platform consists in allocating each functional component onto the hardware resources. Due to the space limitation, we only give two possible allocation scenarios w.r.t the lip synchronization algorithm, and the deployment procedure is also omitted.

**Figure 6** depicts the configuration (C1) that the synchronization component is executed on the processor while the media sources/output devices are all mapped to the slower sensors/actuators. Figure 7 shows the configuration (C2) of the mapping scenario with the synchronization component allocated to the hardware accelerator and all the other components allocated to the faster devices. The mappings from software components to hardware resources are noted by dotted lines in the figures.
C. Step 2: Formal Analysis of System Models

1) Configuration Analysis: Due to space limitations, we illustrate only the analysis of configurations w.r.t the lip synchronization mode concerning their correctness, execution time as well as energy consumptions. The method proposed in [1] is employed for the analysis. For simplicity, a single playable unit is considered for the analysis. We firstly identify temporal properties by defining logical clocks for each component, then synthesize a physical clock for each hardware resource, and at last analyze each configuration by mapping logical clocks to physical ones.

Identification of functional temporal properties. The temporal relations between the components are modeled by using a logical binary clock. An occurrence of 1 means the activation of the component whereas 0 means no activation. And at each activation instant (which corresponds to an instruction cycle), one logical data unit is consumed and produced. Figure 8 depicts the activation traces of logical clocks Vclk, Aclk, Sclk, SCclk and SPclk corresponding to components video, audio, the lip synchronization, screener and speaker.

Synthesis of physical clocks. In order to relate the logical clocks to physical ones, the approach in [1] traces the activation of each hardware resource by computing the period value between two successive processing cycles, e.g. a processor with frequency 30 MHz has the period value equal to approximately 0.033 microseconds, that is 1/(30 MHz). A most frequent clock, called ideal clock, is also defined (by computing the Least Common Multiple) in order to synchronize multi clocks. Figure 9 depicts the ideal clock and different physical clocks (resp. VSen1Clk, ProClk and HwAClk) associated with the hardware resources (resp. VSen1, processor and hardware accelerator) as well as their relations. The physical clocks ASen1Clk, ScAct1Clk and SpAct1Clk (resp. VSen2Clk, ASen2Clk, ScAct2Clk and SpAct2Clk) associated with ASen1, ScAct1 and SpAct1 (resp. VSen2, ASen2, ScAct2 and SpAct2) have the same clock as VSen1Clk (resp. HwAClk).

Analysis of configurations. We firstly consider the configuration (C2) of Figure 6. We assume the number of cycles executed, at the activation instant of components, by hardware resources is 1. By means of mapping the logical clocks onto the physical ones, we get the result shown in Figure 10. The value -1, whose meaning depends on its nearest preceding value that is not -1, means active when it is 1 and idle when 0. As we can see the logical clock properties are not respected. The constraint between VSen1Clk' and ProClk' is violated, because the first activation of ProClk'' happens earlier than the second activation of VSen1Clk'. That is, the video sensor is activated not frequently enough while the processor does not get the data required for its processing.

Then, we consider the configuration (C2) in Figure 7, for which the hardware accelerator is used for the synchronization process whereas both media sensors use the faster frequency value 45 MHz. Figure 11 gives the result of this mapping. This time the temporal properties are respected, and the execution time, which is pointed out by the empty triangle, is 7 ticks (or (7 – 1) × 0.011 microseconds). However, in consideration of
energy consumption minimization, the slack time, defined by the difference of the task deadline (pointed out by the black triangle) and execution time, is big (12 ticks for each hardware resource). Thus, we consider another configuration (C3) that replaces the accelerator and actuators in C2 with the processor and slower actuators and analyze it in the same way (see Figure 12). As a result, it respects the temporal properties, takes 19 ticks for the execution time, and has a relatively small slack time (12 ticks for two sensors and 9 ticks for the processor).

Similarly, all possible configurations of the system can be analyzed, and meaningful information, e.g. correctness, execution time, can be collected.

2) Reconfiguration Analysis: We take into account the previous three configurations C1, C2 and C3 for the LipSync mode and a correct configuration for the SlideShowSync mode C4 to describe our reconfiguration analysis.

Specification of configuration controller. Having a number of possible configurations/modes for the system, the UML Finite State Machine (FSM) is used to model and manage all these configurations as well as their switches. As shown in Figure 13, the FSM specifies how mode values are produced for selecting configurations of the system. It has four states corresponding to the four configurations we have mentioned above. Each state is associated with a specific mode value and the active configuration is the one having the same mode value of the current state of the controller FSM.

We assume that the controller has two inputs: mode switch and the current energy level (High or Not High). As we can see, the designed controller controls that the system chooses the correct mode for the synchronization processing, and in each mode, it behaves according to the current energy level. Controllable variables (i.e. ctr1, ..., ctr6) are also defined in the controller in order to provide controllable points for the DCS tool to enforce system requirements as shown next.

Enforcing system requirements. We firstly encode the designed configuration controller described in Figure 13 into BZR (see Figure 14). Meanwhile, the analysis results for each configuration, i.e. its correctness, execution time and energy consumption, from Section IV-C1 are also associated with each state. In the program, we simply use the amount of ticks of execution time and slack time as the values of the execution time and energy consumption.

<table>
<thead>
<tr>
<th>automation state Lip do</th>
<th>state C3 do correctness = true;</th>
<th>execution time = 19;</th>
<th>energy_consumption = 33;</th>
</tr>
</thead>
<tbody>
<tr>
<td>automaton state C2 do correctness = true;</td>
<td>until not energy_high &amp; ctr6 then C1</td>
<td>not energy_high &amp; ctr3 then C1 end</td>
<td></td>
</tr>
<tr>
<td>execution_time = 7;</td>
<td>energy_consumption = 60;</td>
<td>until switch then Slide</td>
<td></td>
</tr>
<tr>
<td>state C1 do correctness = false;</td>
<td>until energy_high &amp; ctr3 then C2</td>
<td>state Slide do</td>
<td></td>
</tr>
<tr>
<td>execution_time = 19;</td>
<td>energy_consumption = 0;</td>
<td>correctness = true;</td>
<td></td>
</tr>
<tr>
<td>energy_consumption = 46;</td>
<td>until not energy_high &amp; ctr4 then C3 until switch then Lip end;</td>
<td>until switch then Lip</td>
<td></td>
</tr>
</tbody>
</table>

Figure 14. The BZR program for the control automaton

The system constrains are defined by using BZR contracts [10]. Figure 15 gives the code of the contract description of the system requirement that the system always stays away from incorrect configurations, and meanwhile, the execution time constraint, whose limit is defined as a constant (e.g. 19 ticks), must also be respected.

contract var time_constraint:bool; let time_constraint = (execution_time<=19); tel assume true enforce (correctness & time_constraint) with (ctr1,ctr2,ctr3,ctr4,ctr5,ctr6.bool)

Figure 15. The BZR contract enforcing system requirements

At last, by feeding the BZR program and the contract to the BZR compiler, it will synthesize a controller (if it exists) automatically satisfying the system requirement. Figure 16
depicts the simulation results of this case study with respect to different execution time limits (i.e. 19 ticks for the left simulation and 10 ticks for the right one).

Let’s firstly look at the simulation on the left, for which all three configurations satisfy the execution time limit 19 ticks. Initially, the system is in state C2, and when the energy level is low (depicted as energy_high equal to 0 in the figure), the controller inhibits the system from going to state C1, by setting the value of the controllable variable ctr6 to 0, which is an incorrect configuration. The same happens when the system is in state C3. As a result, the system stays in configurations C2 and C3. When the execution time limit is set to 10 ticks, for which only C2 satisfies the execution time limit. As we can see from the simulation on the right, the controller forbids all transitions by setting corresponding controllable variables to 0 and makes the system stay in configuration C2.

D. Step 3: Composition of synthesized controller

Finally, the synthesized controller generated by BZR is integrated into the initial system model (as shown in Figure 17). It gives the values of the controllable variables w.r.t the current system state and input such that the resulting controlled system satisfies the system requirements.

V. CONCLUSIONS

In this paper, we propose a methodology for the safe design of dynamically reconfigurable embedded systems. Such systems are getting an increasing attention due to the crucial needs to address the frequent evolution of embedded systems, regarding requirements from their environments or execution platform in terms of performance and energy consumption. The UML standard profile MARTE has been adopted for the high-level modeling of system configurations (i.e. functionality, execution platform and their allocation). The resulting models are analyzed by using the approach proposed in [1], which employs abstract clocks inspired by the synchronous approach [2], for a fast and qualitative analysis about functional and nonfunctional properties of system configurations via the analysis of scheduling traces resulting from the mapping of logical clocks (capturing functional properties) onto physical ones (derived from hardware processing frequencies). The DCS formal technique is then used for synthesizing a correct controller enforcing reconfiguration correctness. The domain specific language BZR [10] has been adopted for this purpose. At last a CM player case study has been given to illustrate our approach.

REFERENCES

Programming Massively Parallel Architectures using MARTE: a Case Study

A. Wendell Rodrigues                      Frédéric Guyomarc’h                      Jean-Luc Dekeyser
LIFL - USTL                              LIFL - USTL                              LIFL - USTL
INRIA Lille Nord Europe - 59650          INRIA Lille Nord Europe - 59650          INRIA Lille Nord Europe - 59650
Villeneuve d’Ascq - France               Villeneuve d’Ascq - France               Villeneuve d’Ascq - France
{wendell.rodrigues, frederic.guyomarch, jean-luc.dekeyser}@inria.fr

Abstract—Nowadays, several industrial applications are being ported to parallel architectures. These applications take advantage of the potential parallelism provided by multiple core processors. Many-core processors, especially the GPUs(Graphics Processing Unit), have led the race of floating-point performance since 2003. While the performance improvement of general-purpose microprocessors has slowed significantly, the GPUs have continued to improve relentlessly. As of 2009, the ratio between many-core GPUs and multicore CPUs for peak floating-point calculation throughput is about 10 times. However, as parallel programming requires a non-trivial distribution of tasks and data, developers find it hard to implement their applications effectively. Aiming to improve the use of many-core processors, this work presents an case-study using UML and MARTE profile to specify and generate OpenCL code for intensive signal processing applications. Benchmark results show us the viability of the use of MDE approaches to generate GPU applications.

I. INTRODUCTION

Advanced engineering and scientific communities have used parallel programming to solve their large scale complex problems for a long time. Despite the high level knowledge of the developers belonging to these communities, they find hard to implement their parallel applications effectively. Over recent years, using Graphics Processing Units (GPUs) has become increasingly popular and in fact important for seeking performance benefits in computationally intensive parallel applications. However, even for GPUs, there are not performance gains without challenges: first, the identification and exploitation of any parallelism in the application is responsibility of programmers. Often, this requires intimate understanding of the hardware and extensive re-factoring work rather than simple program transformations. Second, the high-level abstractions of a problem can hardly be expressed in the CUDA [2] or OpenCL [3] programming model. Furthermore, subsequent manual optimisations distort any remaining abstractions in the application.

A way of addressing this abstraction is to provide a model-to-source transformation mechanism where the model is captured through a Model-Driven Environment (MDE) [6] and then the code generation is handled by templates. Gaspard2 [5] is a framework that uses UML and the MARTE profile in order to implement its MDE approach. This paper shows a case-study using Gaspard2 to generate an application code for massively parallel architectures based on GPUs. Moreover, performance issues are applied to the model transformation and code generation steps in order to achieve optimization levels accomplished in manually written codes.

II. BACKGROUND

A. Massively Parallel Architectures

GPU is a manycore processor attached to a graphics card dedicated to calculating floating point operations. The GPU devotes more transistors to data processing rather data caching and flow control. This is the reason why the GPU is specialized for compute intensive. Nevertheless, even if GPUs are a manycore processors, their parallelism continues to scale with Moore’s law. It is necessary to develop application software that transparently scales its parallelism. GPUs such as NVIDIA GeForce GTX 480 contain 15 Streaming Multiprocessors, each of which supports up to 1024 co-resident threads, so 30K threads can be created for a certain task. In addition, each multiprocessor executes groups, called warps, of 32 threads simultaneously. NVIDIA’s actual CUDA architecture, code-named Fermi, has features for general-purpose computing. Fundamentally, Fermi processors are still graphics processors, not general-purpose processors. The system still needs a host CPU to run the operating system, supervise the GPU, provide access to main memory, present a user interface, and perform everyday tasks that have little or no data-level parallelism.

B. OpenCL

OpenCL(Computing Language) is a standard for parallel computing consisting of a language, API, libraries and a runtime system. Originally, was proposed by Apple, and then turned over to the Khronos Group.

OpenCL also defines a programming language for writing kernels, which is an extension of C. Kernels are executed within their own memory domain and may not directly access host main memory. OpenCL usually defines a host where main programs are placed and one or more devices that executes kernels. Furthermore, device memory is divided into four distinct regions:

- Global memory, a kind of ”device main memory”, can be accessed by all work-items and the host in reads/writes.
- Constant memory is similar to global memory, except that work-items may only read from this memory.
- Local memory is read/write memory local to a workgroup, and is shared by all work-items of this group.
- Private memory is local to each work-item.

The OpenCL programming language defines type qualifiers to specify in which memory region a variable is stored or a pointer points to. As a kernel can neither access host main memory nor dynamically allocate global and constant memory, all memory management must be done by the host. The OpenCL API provides functions to allocate linear memory blocks in global or constant memory, as well as to copy data to or from these blocks.

C. MARTE in Gaspard2 Context

MARTE (Modeling and Analysis of Real-Time and Embedded systems) [1] is a standard proposal of the Object Management Group (OMG). The primary aim of MARTE is to add capabilities to UML for model-driven engineering of real-time and embedded systems. UML provides the framework into which needed concepts are plugged. The MARTE profile enhances possibility to model software, hardware and relations between them. It also provides extensions to make performance and scheduling analysis and to take into account platform services. Gaspard2 [5] is a framework based on MDE and MARTE profile. From a high-level abstraction model of application, architecture and allocation, Gaspard2 provides transformation chains and templates to code generation for several target platforms. One of these platforms is the hybrid (CPU+GPU) platform beneath the OpenCL API.

III. Case Study: H.263 Video Downscaling

The case study concerned in this paper deals with specific aspect of H.263-based video compression standard, scaling. The scaling during video-compression is considerably important for previews or for streaming for small form factor devices, such as mobile phones. The application consists of a classical downscaler, which transforms a video signal, which, for instance, is expressed in Common Intermediate Format (CIF), into a smaller size video. In this situation, the downscaler can be composed of two components: a horizontal filter that reduces the number of pixels from 352-lines to 132-lines and a vertical filter that reduces the number of pixels from 288-lines to 128-lines by interpolating packets of 8 pixels both row- and column-wise.

In a typical case of handling a 25-frames-per-second video signal lasting for 80-seconds, the downscaler may process up to 2000 frames in CIF format, with each input frame being represented by a two-dimensional array of size 352 × 288 and should emit 2000 output frames of size 132 × 128. Since each video pixels is encoded in 24-bit RGB colour model, the frame generation process is repeated for each frame and for each pixel of different colour space along two different directions. The final frame is produced by using these outputs from different colour space. Depending on the composing function, a broad-range of output colours are possible for each pixel and thus for each frame. The figure 1 illustrates this basic operation for a given frame in high-definition format.

As can be observed, the operations concerned with the scaling is highly parallel and repetitive. The interpolation is repeated for each frame, each pixel and for each colour channel.

A. Downscaler Model

The figure 2 gives us an overview of the downscaler application. The figure illustrates a model for a 300-frames video, even if we have 2000 frames in our testbed. For this example, we are going to analyse only the first repetitive task from the Horizontal Filter component. The other tasks have equivalent behaviour. The yhfk task has a multiplicity equals to [288,44]. It means this task is composed of 288x44 independent tasks (so-called Elementary Task), and thus, parallelizable. Each elementary task takes a pattern from the input array. A tiler stereotype1 do the tiling operation. It allows to split input data in patterns in accordance with tiler’s array definitions of origin, paving and fitting. Besides features such as tiler specifications and task repetitions, MARTE profile is applied to OpenCL architecture definition (host and device) in order to grant task allocations. For this illustrated model, we add a specification to one host (CPU+Memory) and one device (GPU+Global Memory). Data and tasks are placed into memories and processors according to project interests. For instance, the six repetitive tasks in horizontal and vertical filters are allocated onto the GPU in order to generate kernels in the execution environment. Allocate stereotypes are used to map ports and tasks into HwRAMs and HwProcessors. These stereotypes will allow for creating all variables and relations between them. Additionally, in order to distinguish host from device, we modify the description attribute from HwResource stereotype.

B. MARTE to OpenCL Transformation Chain

Gaspard2 supplies a transformation engine that allows us chaining a set of model-to-model or model-to-text transformations. These transformations take into account model elements and properties and gather information to create cleaner models towards a target platform. A recent chain was added to Gaspard2 framework and it allows for automatic code generation from MARTE to OpenCL (see [6]). Subsequent paragraphs highlight some details of the designed model taken into account by these transformations.

1Defined in ArrayOL [4] language and part of MARTE.
1) Launch Topology: An allocated repetitive task should be properly executed. In OpenCL programming model, elementary tasks are work-items in a work-group context. The work-group and work-item topology (grid of threads) are computed from multiplicity of the elementary task. For instance, a MxN multiplicity is transformed in the work-item topology as defined in the figure 3. Threshold levels help to avoid mistaken topology definitions for smaller or bigger multiplicities.

2) Data Allocation: A critical problem in application modeling based on MDE is to manage the data allocation in the target platform. MARTE profile adds the flowPort stereotype to UML port element. The main attribute aggregated to a port element is the direction, which allows to define whether the port is input, output, or bidirectional. This information contributes to decide which elements are read-only variables.

By using UML links we can associate flowPorts to memories in architecture models. Each port has attributes and associations that allow us defining size and data type for example. Thus, developers can specify in their application models where the data will be stored and how much space will occupy the data. In the figure 4 we can see a simple example of allocation. Ports from different tasks are allocated into memory elements of their respective processors.

3) Performance Tuning: Usually connected ports allocated to different memory boxes, as seen in the data allocations illustration (figure 4), cause a data transfer between CPU and GPU. At a first sight, one can say that is a critical point to decrease the performance because subsequent kernels reuse these data. Since unnecessary data transfer times are expressive in running time, it would be interesting to take excessive transfers in the model design. Nevertheless, applying some intelligence levels to the transformations we can detect these critical points in the original model and avoid extra data transfers. Therefore, performance gains, as observed in result charts in next section, can be achieved automatically.

IV. RESULTS

Four versions of the Downscaler were tested. The first one is a sequential version using the same structure defined in the figure 2. The other two versions are OpenCL automatically generated and the last one is a manually written OpenCL version. We have used a transformation chain that transforms model to model using QVTO [8] and model to text using Acceleo [7]. The first OpenCL code is a not optimized program without any further analysis on memory transfers. The second one regards the memory transfers between host and device. Minimize these transfers reduces notably the total GPU execution time. As seen in the figure 5, data transfers take a lot of time (more than 70%) in the Downscaler application. The communication takes more time than computing process by the work-items. Time analysis in figure 5 demonstrates the bigger spent time in y-component kernels due to their bigger topology and handled data. After the performance tuning, no time changes occurs in kernels (as it was expected). However, we verify about 30% and 70% faster transfer times in host to device and device to host respectively.

The figure 6 presents the total execution time of each implemented version. Both OpenCL codes give us good results with relation to CPU code. Using optimized transfers we can achieve speedups of 10x. In fact, structurally, the optimized version is really closer to manually written (considering the model designer is the code programmer). The decision of the topology and data transfers by the transformations (model compiler) were closely inspired by decisions taken if they would a human programmer. For the optimized version we achieve about 25% of speed-up. This is an enough expressive performance for two parallel implementations.

V. CONCLUSION

Even though we have used an application model not specially developed for GPU architectures, we have had good results at performance level. Parallel languages are hard to
program and MDE approaches are well suitable to allow not specialized programmers creating parallel programs. The results presented in this work help us to certify the high potential of MARTE profile to create parallel applications for massively parallel processors. For the time being, we are implementing more optimization features in the transformation chain in order to ensure a stable and generic framework to create OpenCL applications exploiting, among other things, the memory hierarchy throughput.

REFERENCES

A Retargetable SysML-based Front-End for High-Level Synthesis

Fabian Mischkalla, Da He, Wolfgang Mueller
University of Paderborn/C-LAB
Paderborn, Germany

Florent Azcarate, Manuel Carballeda
Thales Security Solutions & Services S.A.S
Velizy Villacoublay, France

Abstract—UML profiles like SysML and MARTE have been a major research topic in electronic system design, but are mainly applied for specification and analysis in early design phases. High-Level Synthesis (HLS), however, addresses the physical implementation aspect of electronic systems, and thus leads to different requirements on the accuracy of models. For this, modular interfaces are a novel object-oriented synthesizable technique to overcome the conflict between a higher degree of abstraction and necessary details for further synthesis.

In this paper, we present our approach to use SysML as an adequate modeling language for modular interfaces and C/C++/SystemC-based HLS. We extended SysML with annotations for synthesizable SystemC and high-level synthesis constraints and implemented a code generation scheme to achieve design flow automation. Based on the SysML editor Artisan Studio and an industrial case study, we demonstrate the applicability of SysML as a retargetable front-end for HLS design flows.

Index Terms—UML, SysML, SystemC, High-Level Synthesis

I. INTRODUCTION

UML for electronic system design has been studied now for almost a decade [1]. Thus, there exist modeling standards like SysML [2] and MARTE [3] that are adequate for systems modeling and analysis mainly for early design steps. High-level synthesis (HLS) research has its beginnings in the 1980s and achieves now an industrial applicable technology to meet the increasing time-to-market demands for digital systems [4].

This paper introduces our approach to provide a retargetable and automated design flow from SysML to behavioral synthesis (see Figure 1) as one part of our overall HW/SW co-design methodology [5]. We augmented an advanced SysML environment with the support of high-level synthesis constraints to integrate generic C/C++/SystemC-based HLS design flows. We generate synthesizable SystemC code as an intermediate language for simulation as well as for synthesis. Additionally, SystemC testbenches are considered for equivalence checks of the SystemC behavioral input and the synthesized RTL output. We also derive configuration files and makefiles directly from the model allowing one-click design flow automation.

During the evaluation phase it has been shown that for efficient application of SysML targeting HLS requirements, it is essential to reduce the tedious overhead of pin-accurate interface modeling. For this, we developed new extensions for modeling modular interfaces in order to abstract the complex wiring of ports and signals by a synthesizable object-oriented mechanism. We redesigned a complex industrial case study previously implemented in [6], generated the SystemC executable and further synthesized it for FPGA configuration. Our experiments have shown that we reduced the typical interface modeling overhead in SysML of up to 70%.

The remainder of this paper is structured as follows. In the next section we explain our technical contribution in the context of previous work. Section III explains SystemC code generation including the application of modular interfaces. Section IV describes our high-level synthesis extensions offering designers the control over synthesis at modeling time. An overview of our evaluation scenario is presented in section V before section VI ends up with a conclusion.

II. RELATED WORK

Since the introduction of UML2 profiles many initiatives are working on the adaptation of UML for SystemC-based Systems on a Chip (SoC) design. We can identify two main directions: One aims on precise model-to-code transformations for simulation and implementation, and the other one has a rather abstract intent for analysis and allocation.
A comprehensive overview of several model-to-code relationships can be found in [7]. In fact, our framework follows their one-to-one mapping approach for efficiently capturing the semantics of the underlying synthesis subset and allowing round-trip capabilities by common UML tools.

A first investigation on combining UML and SystemC in a one-to-one relationship was done in [8], where they describe several benefits especially at early stages of SoC design. Besides further academic work [9][10][11][12] for direct code generation as well as for code generation from the XML Metadata Interchange (XMI) format, there were also industrial efforts. In this context, Fujitsu pushed the development of the OMG UML Profile for SoC [13], and STMicroelectronics introduced their UML for SystemC profile [14] soon after. All of these approaches address the generation of SystemC and map SystemC semantic directly to UML. Although adopting some concepts from these profiles, our approach is more low-level oriented in order to cover high-level synthesis requirements and be applicable for complex systems with a huge amount of wiring. For instance, in contrast to the abstract process modeling in [13] we classify process behavior into sequential and combinatorial logic as described in [15]. However, we avoid the explicit modeling of internal behavior as proposed in [14] because it turned out that writing plain SystemC code at this point is more convenient. Finally, we consider synthesizable SystemC and provide efficient modeling utilities for modular interfaces via metaports as they are not precisely relatable to the standard port/interface concept of UML. In detail metaports are hierarchical ports, which consist again of ports, signals, processes, whereas UML standard ports and provided/required interfaces are based on function call/implementation.

In contrast to the one-to-one mapping approaches, there are OMG standards like MARTE intended to be more general and language independent. For this, MARTE targets rather the functional view of hardware components and does not consider implementation and synthesis details, which we address by our extensions. Nevertheless as the semantic of the extensions is not limited to SysML, so that MARTE could be applied as well, for instance.

Some additional efforts have been spent to combine both directions. For this, in [16][17] they bridge the gap between UML and IP-XACT [18], the de-facto standard for IP integration and management. But as IP-XACT does not explicitly consider high-level synthesis constraints as well, we see our extensions here similarly as a well-suited add-on.

### III. SYSTEMC CODE GENERATION AND MODULAR INTERFACES

Our code generation scheme is implemented in Studio’s template-based code generator language and is based on SysML due to its tight integration into the tool. For this, table 1 summarizes our transformation rules from SysML to synthesizable SystemC. The SysML structure can be modeled by three types of SysML diagrams. System engineers should use Block Definition Diagrams (BDD) for system components and their composition. The internal system and communication structure is represented by Internal Block Diagrams (IBD). With SysML Activity Diagrams (AD) one models the SystemC process alignment in terms of sensitivity and port access.

Typical SystemC HLS tools have restrictions on their supported input language constructs. As such synthesizable SystemC modules have to be designed with signal-level ports, which lead to a high modeling overhead if SysML is used in the same way as a Hardware Description Language (HDL). For this, modular interfaces [19] are a novel synthesis technique for hiding recurring pin-accurate interface details from the designer, but still providing them in the model for simulation and synthesis. By means of object-oriented capabilities individual signals and ports are encapsulated into reusable classes and connected with only one binding call. So each modular interface describes a specific communication protocol (e.g. memory access or bus interface) and consists of two complementary sockets, i.e., metaports, as well as an intermediate channel. The communication between a component and his socket is provided by the transaction-level API defined in the socket class. We extend SysML to adapt this mechanism for archiving the following advantages:

- Reducing exceedingly the structural modeling effort in graphical representations of synthesizable SysML models.
- Providing reuse of standard interfaces among multiple design alternatives via model libraries.
- Avoiding redundant development time by strict separation of behavioral and communication sources.

<table>
<thead>
<tr>
<th>SysML Element</th>
<th>SystemC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block</td>
<td>sc_module/sc_clock</td>
</tr>
<tr>
<td>BlockProperty</td>
<td>Attribute(C++ Instance)/sc_signal/sc_fifo</td>
</tr>
<tr>
<td>StandardPort</td>
<td>sc_port</td>
</tr>
<tr>
<td>FlowPort</td>
<td>sc_in/sc_out/sc_inout</td>
</tr>
<tr>
<td>Connector</td>
<td>Port Binding</td>
</tr>
<tr>
<td>Activity</td>
<td>Static Behavior Structure</td>
</tr>
<tr>
<td>Action Node</td>
<td>Sequential/combinatorial Process</td>
</tr>
<tr>
<td>Operation</td>
<td>Process Behavior</td>
</tr>
<tr>
<td>Pin/Parameter</td>
<td>Process Sensitivity</td>
</tr>
</tbody>
</table>

Table 1: SystemC code transformation rules

Figure 2: SysML IBD of a modular interface
As shown in Figure 2, we specify synthesizable modular interfaces labeled with "<<ModularIO>>" by standard SysML objects like blocks, reference properties (specified by dotted lines) and flow ports. As such, the ambient block specifies a channel and the reference properties reflect his associated metaports. From this model our code generator generates a header file composed of three classes (Listing 2). The class DVI_P2P defines a channel including all low-level signals. The two remainder classes, DVI_Initiator and DVI_Target, represent passive metaports, which contain opposed port interfaces, binding functions, and the transaction API. However, a metaport can also be active, i.e., owning multiple SystemC processes and being derived from sc_module.

The application of modular interfaces is done by SysML allocations (Figure 3). For this, a "<<Metaport>>" stereotype defined in our extensions denotes a SysML standard port as a metaport, which can be allocated from a SysML block for the implementation. A connector between metaports indicate either a high level channel if allocated from a channel implementation or a hierarchical delegation through metaports of the same type. The right allocation of correspondent port/channel types is automatically ensured by means of constraints checking via VBScript in Artisan Studio. The generated SystemC code for the structure of Figure 3 is shown in extracts in Listing 3.

IV. SYSML EXTENSIONS FOR HIGH-LEVEL SYNTHESIS

Additionally to synthesizable SystemC code generation for verification, the framework targets HLS to avoid the error-prone manual refinement from behavioral-level SystemC to hardware implementation. For SystemC-based synthesis tools like the SC Compiler from Agility, SystemCrafter SC, Forte Cynthesizer or CatapultC from Mentor Graphics are qualified. Due to their support of additional high-level synthesis
and updates the running script callable from the Artisan Studio tool dependent synthesis constraints, new configuration files . The code generator generates instantaneously by simply redefining the different compilers, a designer can retarget the code generator underly ing SysML modeling style. For switching between two SystemCrafter v3.0 without any modifications on the underlying SysML specification to retarget code generation and control further synthesis. As such, the stereotypes in table 2 address these high-level synthesis directives, which can be grouped into structural and behavioral ones.

For structural constraints as similar to simulation an engineer must be able to specify high-level synthesis entry points. For this, <<HLS TopLevel>> denotes a SysML block as top-level and provides means to define the targeting synthesis compiler. Afterwards subjacent components annotated by <<HLS Blackbox>> are considered to be library modules with only instantiation in the synthesized RTL. For specifying modular interfaces as mentioned before, we define <<ModularIO>> and <<Metaport>> classified as passive or active socket implementation.

In case of HLS behavioral directives, it is required to explicitly specify the reset behavior of internal logic, e.g. registers and outputs. Therefore, <<HLS GlobalReset>> on a SystemC sc_in<bool> port determines reset ports and forces automatically the generation of the SystemC reset logic, i.e., a reset_signal_is and sensitive statement for SC_CTHREAD and SC_METHOD, respectively. Furthermore, memories in abstract functional specifications are typically described in form of C++ arrays. For cycle-accurate timing verification and synthesis, it is important to map these arrays to platform implementation models, e.g., flattened registers or RAM/ROM cells, which are supported by <<HLS Memory>>. Finally, <<HLS External>> on SysML elements provides the capability to generate conditional debug and synthesis code. Such code is then included in preprocessing statement specified by a directive and an optional argument, e.g. the targeted synthesis subset. Thus, an argument of SC_SYNTHESIS >= 0x123 denotes a synthesis subset version of 1.23.

Our approach currently supports SC Compiler v1.3 as well as SystemCrafter v3.0 without any modifications on the underlying SysML modeling style. For switching between two different compilers, a designer can retarget the code generator by simply redefining the compiler property of <<HLS TopLevel>>. The code generator generates instantaneously tool dependent synthesis constraints, new configuration files and updates the running script callable from the Artisan Studio GUI. The generated SystemC code labels the synthesis directives by a set of C/C++ preprocessing statements. By SC_SYNTHESIS or self-defined arguments, code pieces, that are helpful for debugging and simulation, but which shall not or cannot be synthesized, can be switched off for synthesis as illustrated in listing 3. After code generation, the HLS process can be started by customized context menus.

<table>
<thead>
<tr>
<th>SysML Element</th>
<th>Stereotype</th>
<th>Tagged Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block</td>
<td>&lt;&lt;HLS TopLevel&gt;&gt;</td>
<td>+ compiler</td>
</tr>
<tr>
<td>Block</td>
<td>&lt;&lt;HLS Blackbox&gt;&gt;</td>
<td>+ source</td>
</tr>
<tr>
<td>Block</td>
<td>&lt;&lt;ModularIO&gt;&gt;</td>
<td>+initiator +target</td>
</tr>
<tr>
<td>StandardPort</td>
<td>&lt;&lt;Metaport&gt;&gt;</td>
<td>+type</td>
</tr>
<tr>
<td>FlowPort</td>
<td>&lt;&lt;HLS GlobalReset&gt;&gt;</td>
<td>+ polarity</td>
</tr>
<tr>
<td>BlockProperty</td>
<td>&lt;&lt;HLS Memory&gt;&gt;</td>
<td>+memType +isAsync</td>
</tr>
<tr>
<td>All</td>
<td>&lt;&lt;HLS External&gt;&gt;</td>
<td>+directive +argument</td>
</tr>
</tbody>
</table>

Table 2: HLS extensions for SysML

directives, the tools provide us with the opportunity to extend the SysML specification to retarget code generation and control further synthesis. As such, the stereotypes in table 2 address these high-level synthesis directives, which can be grouped into structural and behavioral ones.

For structural constraints as similar to simulation an engineer must be able to specify high-level synthesis entry points. For this, <<HLS TopLevel>> denotes a SysML block as top-level and provides means to define the targeting synthesis compiler. Afterwards subjacent components annotated by <<HLS Blackbox>> are considered to be library modules with only instantiation in the synthesized RTL. For specifying modular interfaces as mentioned before, we define <<ModularIO>> and <<Metaport>> classified as passive or active socket implementation.

In case of HLS behavioral directives, it is required to explicitly specify the reset behavior of internal logic, e.g. registers and outputs. Therefore, <<HLS GlobalReset>> on a SystemC sc_in<bool> port determines reset ports and forces automatically the generation of the SystemC reset logic, i.e., a reset_signal_is and sensitive statement for SC_CTHREAD and SC_METHOD, respectively. Furthermore, memories in abstract functional specifications are typically described in form of C++ arrays. For cycle-accurate timing verification and synthesis, it is important to map these arrays to platform implementation models, e.g., flattened registers or RAM/ROM cells, which are supported by <<HLS Memory>>. Finally, <<HLS External>> on SysML elements provides the capability to generate conditional debug and synthesis code. Such code is then included in preprocessing statement specified by a directive and an optional argument, e.g. the targeted synthesis subset. Thus, an argument of SC_SYNTHESIS >= 0x123 denotes a synthesis subset version of 1.23.

Our approach currently supports SC Compiler v1.3 as well as SystemCrafter v3.0 without any modifications on the underlying SysML modeling style. For switching between two different compilers, a designer can retarget the code generator by simply redefining the compiler property of <<HLS TopLevel>>. The code generator generates instantaneously tool dependent synthesis constraints, new configuration files and updates the running script callable from the Artisan Studio GUI. The generated SystemC code labels the synthesis directives by a set of C/C++ preprocessing statements. By SC_SYNTHESIS or self-defined arguments, code pieces, that are helpful for debugging and simulation, but which shall not or cannot be synthesized, can be switched off for synthesis as illustrated in listing 3. After code generation, the HLS process can be started by customized context menus.

<table>
<thead>
<tr>
<th>#ifdef SC_SYNTHESIS &gt;= 0x123</th>
</tr>
</thead>
<tbody>
<tr>
<td>… // code for subset 1.23 or later</td>
</tr>
<tr>
<td>#endif</td>
</tr>
</tbody>
</table>

Listing 3: Conditional SystemC code generation

V. EVALUATION

For evaluation, we applied a complex industrial smart camera system for Automatic License Plate Recognition (ALPR), which is an image-processing technology used to identify vehicles by their license plates. The original platform implementation without modular interfaces was undertaken by Thales Security Solutions and Services in a joint European project [6] and was completely modeled in Artisan Studio using our HW/SW co-design methodology from [5]. After SystemC code generation, synthesis to RTL was done by the Agility Compiler. Furthermore, they generate the bitstream with Xilinx ISE toolchain and upload it to a Virtex-5 FPGA integrated into their ML507 Development Platform for prototyping.

The smart camera system is built in multiple pipeline stages of combined hardware and software components. The abstract functionality can be roughly divided into the three following phases:

- Image Acquisition & Camera Control (HW & SW)
- Image Processing & Labeling (HW & SW)
- Optical Character Recognition(OCR) Functionality(SW)

The first phase is in charge of capturing the pixel stream and setting up the camera by means of exposure time and gain computation calculated on the grabbed images. During the image processing and labeling phase several filters in form of mathematical morphological operations are passed through, which performs the base algorithms of license plate detection. These filters are applied to detect regions containing a high density of small elements, which is a well-suited characteristic of text like on license plates. The top-level structure of the developed image processor is shown in Figure 4. It is the main core of this phase and can be launched in the following different variants.

- Basic dilation/erosion operators with handling of variable window size.
- Composition of dilation and erosion for opening and closing filters
- Top Hat algorithms based on opening and closing with subtraction of the original image
Afterwards, the identified regions are passed to the OCR application running on a PowerPC440 to read the actual plate number or discard non-text.

For demonstrating the benefits of modular interfaces modeling and retargetable code generation, we focused on the morphological filter communication infrastructure as one part of the smart camera system and redesigned the entire image processor. However, HW/SW interfaces, in our case the IBM CoreConnect architecture, are also an interesting area of application for modular interfaces. An overview of the communication structure is shown in Table 3.

<table>
<thead>
<tr>
<th></th>
<th>SysML Blocks</th>
<th>SysML Ports</th>
<th>SysML Connectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Model</td>
<td>20</td>
<td>292</td>
<td>166</td>
</tr>
<tr>
<td>Model using Modular Interfaces (without Reuse)</td>
<td>44</td>
<td>157 (46%)</td>
<td>99 (40%)</td>
</tr>
<tr>
<td>Model using Modular Interfaces (with Reuse)</td>
<td>20</td>
<td>69 (76%)</td>
<td>55 (66%)</td>
</tr>
<tr>
<td>Modular Interfaces</td>
<td>24(8x3)</td>
<td>88</td>
<td>44</td>
</tr>
</tbody>
</table>

**Table 3: Communication structure reduction of the ALPR image processor design**

The original design from Thales is composed of 20 SysML blocks whereof 15 are taken into account for synthesis and 5 for verification purposes. For modeling the communication structure they required a total of 292 SysML flowports and modeled 166 port bindings in form of SysML connectors. With our new extensions we create 8 additional modular interfaces as summarized in the last row and integrate them into the model. Afterwards, the image processor diagram corresponds to Figure 5, which fits much more in the usual UML/SysML modeling appreciation, i.e., analysis and documentation friendly. Taking the implementation of modular interface as new effort (without Reuse), we reached a reduction of 46%. If modular interfaces considered as coming from a preexisting model library, the reduction increases up to 76%. Thus using our extensions engineers could expect an overhead modeling reduction close to 70% after their first design, which is an efficient improvement in designing synthesizable SysML models.

**VI. CONCLUSION**

In this paper, we introduced our retargetable and automated hardware design flow from SysML modeling to SystemC-based HLS. We enhanced Artisan Studios SysML modeling environment so that engineers could efficiently specify and control hardware synthesis via modeled high-level directives. Due to the fact that SysML comes with the default configuration of Artisan Studio, we defined our stereotypes in an additional profile package on top of SysML, which can be simply loaded into the tool. Based on dedicated transformation rules, we implemented a retargeable code generator for synthesizable SystemC as well as design flow automation. The code generation itself executes by permanently observing modifications of the model and updates the code on demand. Via configuration files and preprocessing directives the generated SystemC code is efficiently retargetable for multiple HLS compilers. For evaluation we partly redesigned an industrial smart camera system, which passed the whole design flow from SysML modeling, over SystemC code generation to synthesis. Our measurements show a reduction of the communication structure modeling overhead by almost 2/3.
REFERENCES

Towards SystemC Code Generation from UML/MARTE Concurrent System-Level Models

P. Peñil, F. Herrera & E.Villar
University of Cantabria
E.T.S.I.I.T., TEISA Dpt. Av. Los Castros s/n, Santander (Spain)
{pablop, fherrera, evillar}@teisa.unican.es

Abstract
MARTE is an advanced UML profile which can be used for specifying system-level models, the input for Electronic System-Level (ESL) design methodologies. The validation of these system models through an executable model is necessary to avoid the high costs of propagating system-level specification errors to further design steps. This paper contributes to the automation of the generation of a SystemC executable from a UML/MARTE system-level specification, which captures the concurrency and communication structure of the system. Specifically, the paper focuses on the code generation from Kahn Process Networks in UML/MARTE. Moreover, the paper deals with the code generation of the behaviour of each concurrent element, since it is relevant for the executive semantics. This contributes to the coherence between the MARTE specification and the generated SystemC executable.

I. INTRODUCTION
The design of embedded systems is growing in complexity. Embedded applications include more functionality everyday, while the evolution of electronic technology enables the integration of more computational resources. In this context, Model Driven Architecture (MDA) provides a solution by enabling the description of a Platform Independent Model (PIM), which abstracts the application from the platform details. The Platform Definition Model (PDM), and the way the PIM is mapped onto the PDM, to produce a Platform Specific Model (PSM), is left for later design steps.

In a previous work, a formally supported methodology for the creation of PIMs in UML/MARTE that enables the validation of the system functionality and of its concurrent partition was proposed [1]. These UML/MARTE specifications capture the concurrency and communication (C&C) structure of the system, determining the concurrent elements and how they transfer data and synchronize among themselves. The explicit identification of the concurrent elements facilitates the allocation of the system application to platforms with multiple processing elements in later design phases. Moreover, the proposed methodology also enables the capture of the structure of the behaviour of concurrent elements. Concurrent element behaviour interlaces pure functionality with accesses to communication elements. The structure of the behaviour of a concurrent element specifies how pure functionality and communication accesses are interlaced. Therefore, this structure is as relevant as the concurrency and communication structure, since both are involved in the executive semantics of the process network.

![Figure 1. Code Generation from the UML/MARTE PIM.](image)

This paper focuses on the automatic generation of a corresponding SystemC executable from PIMs under the aforementioned UML/MARTE methodology. Specifically, the paper addresses the generation from UML/MARTE specifications under the Kahn Process Networks (KPN) formalism [2], which provides a formal guarantee of functional determinism. Variants close to KPN, such as Bounded KPN (BKPN) are also supported. This code generation provides an internal system structure composed of the internal subsystem, resulting from system modularization, and the different concurrent agents and communication mechanism that specify the C&C.

For the generation of the SystemC executable model, a set of mapping rules are given. Some rules, introduced in [3], cover the translation of the C&C structure. This paper shows an additional set of rules for the translation of the behaviour structure of each concurrent element, based on the formal link introduced in [1]. This contribution will facilitate the coherence in terms of semantics between the UML/MARTE specification and its corresponding SystemC executable. The result of the corresponding code generation
is a behaviour structure where different code structures such as conditional statements and infinite loops are included.

Moreover, the paper introduces the implementation of a generator prototype. The compilation of the generated code, plus the pure functionality provided by C/C++ library and the SystemC library, produces the executable specification.

II. RELATED WORK

The application scope of UML has been extended to cover different domains from its initial application domain such as object-oriented software system modeling. In this context, in [8] the capabilities of the application for the design of electronic systems are described. Specifically, in order to exploit the benefits of UML as a modelling language, the MARTE profile was created to deal with the modelling of real-time embedded systems.

Some works have dealt with the code generation from MARTE models. In [13] the method for generating Java and C++ code from the SRM MARTE sub-profile is studied. Annotations defined by the MARTE sub-profile for Software Resource Modelling (SRM) provide high-level abstractions of target software platforms, which are used to support the deployment of the application model for various different targets. This work considers that the mapping of an application onto a platform is not only characterized by the API of the targeted OS, but it is also influenced by the design restrictions set by the programming languages and their related libraries. For this, the Platform Specific Models (PSM) either in Java or in C+/POSIX are produced by applying dedicated model transformations to the high-level application model (HLAM), platform independent (PIM), model, by using information provided by the SRM annotations.

Gaspard2 [10] is a design environment dedicated to Multi-Processor Systems-on-Chip (MPSoC) for moving from the high-level hardware architectural description in MARTE to a SystemC executable platform for data-intensive applications. Through model transformations, Gaspard2 enables the generation of an executable TLM SystemC platform at the timed programmers view (PVT) level, thus providing a bridge among several analysis tools.

MoPCoM [9] is a UML/MDA co-design methodology oriented to the design of high-quality real-time embedded systems. MoPCoM uses the MARTE profile as an UML extension to describe real-time properties and to perform platform modelling. MoPCoM defines three abstraction levels. A first Abstract Modelling Level (AML) is used to model system behaviour. It is based on SysML, enhanced with some MARTE elements. A second and intermediate Execution Modelling Level (EML) is used for performance analysis. Finally, a third level called Detailed Modelling Level (DML) allows the automatic generation of implementation code. References [11] and [12] show how MoPCoM enables the automatic generation of implementation code from the DML model. MoPCoM relies on a subset of C/C++ to provide a language for carrying out actions, which after an additional parsing, enables the generation of synthesizable VHDL. MoPCoM is capable of targeting any implementation language; for instance, by extending the action code to support syntax constructs supported by C/C++ based high level synthesis tools.

Both Gaspard2 and MoPCoM are design methodologies that deal with designing the system after architectural mapping, that is, a system where the HW-SW elements are specified. The approach presented in this paper is at a higher level of system design, where the designer identifies the concurrent agents and the required communication mechanisms to connect them. Additionally, in MoPCoM the semantics of different models of computation are specified by means of a set of stereotypes created ad hoc, specifically for capturing the KPN semantics. In this article, we capture the KPN semantics by using MARTE elements. Regarding the code generation, although in Gaspard2 the target language is SystemC, it is oriented to simulation of HW-SW systems. In MoPCoM, the initial target code is VHSL, but this does not imply any restriction for other languages (i.e SystemC). Apart from the code generation of the system structure the methodology presented in this paper enables the automatic generation of the behaviour structure, that is, the skeleton of the functionality associated with each concurrent element. Finally, in order to provide consistency in the UML/MARTE-SystemC and thus, in the code generation, formal foundations are used.

III. UML/MARTE METHODOLOGY

The model-based methodology deals with the modelling of the system’s internal hierarchy and the functionality. By means of Composite Structure diagrams, designers capture the internal structure focusing on the concurrent and communicating aspects. In order to model the functionality implemented by each concurrent agent, Activity diagrams are used. This is shown in Figure 2.

A. C&C modelling

The user captures the C&C structure by means of a Composite Structure diagram, where the basic modelling elements are two stereotypes of the Generic Resource Modelling MARTE sub-profile, namely the ConcurrencyResource and the CommunicationMedia. The ConcurrencyResource is the most abstract element that is able to perform its execution flow concurrently with others. The CommunicationMedia is defined as the generic resource responsible for the transport of information and the synchronization among the concurrency resources.
In the KPN and BKPN cases, a CommunicationMedia must reflect specific communication semantics: a unidirectional FIFO data flow with read and write blocking semantics. To do this, the CommunicationMedia has two Flow Ports, whose direction attributes define the read and write accesses. The StorageResource stereotype is applied to the CommunicationMedia, in order to reflect the FIFO buffering capacity. If the inherited resMult attribute is undefined, an unbounded buffering capacity is assumed, thus covering the KPN case. Otherwise, the bounded buffering capacity supports the BKPN case.

### B. Functionality modelling

The structure of the behaviour of each ConcurrencyResource is modelled by means of an Activity Diagram. In this activity diagram, the communication accesses are modelled by means of the AcceptEventAction and the SendObjectAction UML actions. Both SendSignalAction and AcceptEventAction represent the invocation of a specific service offered by a communication media. In order to avoid inconsistent calls to these services, each one of these services has to be univocally identified in the activity diagrams. For that purpose, each SendSignalAction or AcceptEventAction is clearly identified by name declaration name_communicationMedia.name_Service. Furthermore, to completely remove the inconsistent calls, it is essential to impose a constraint on the service invocation; each concurrency resource can only call the services that are provided by the communication endpoints to which the concurrency resource is connected. The multiplicity of these accesses can also be captured, to denote how many elements are transferred in the action.

Execution of pure functionality is captured as atomic actions (depicted as rounded edge squares in the diagram). These atomic actions represent the individual functions that compose the complete concurrency resource functionality. The functions can correspond to functions provided by external C/C++ libraries, where the pure code implementation of each function is allocated. In other cases, these actions denote the function to be implemented in a later design step according to a description attached to this action.

The activity diagram states, graphically and without any ambiguity, the order of execution among atomic actions (pure functionality) and the aforementioned communication actions. At the same time it is sufficiently abstract to avoid stating coding details which are not semantically relevant, such as which branch of a conditional statement has to appear first in the code.

### IV. SystemC Generation From UML/MARTE

The concurrency resources and communication media are mapped to SystemC processes and channels respectively. Specifically, SC threads are generated from concurrency resources and SC channels are obtained from communication media. Depending on the communication semantics captured in the communication media, a specific SystemC channel is instantiated. In the BKPN case, that is, provided that UML CommunicationMedia is specified as explained in the previous section, when resMult is defined, a standard sc_fifo channel is inferred. When resMult is undefined (KPN case), then an uc_inf_fifo channel, provided by the HetSC library [5] is inferred.

The generation also maintains the component hierarchy of the MARTE specification. In this model-based methodology, the internal hierarchy of the system is captured by UML Components and Ports. SystemC modules and ports are inferred from these UML elements.

The body of the corresponding function declared as a SystemC thread is inferred from each activity diagram associated with a concurrency resource. The generated code reflects the paths of the diagram. It is possible to distinguish different path structures. Figure 3 shows the loop structure and the conditional structure.

**Figure 3 Loop structure and conditional structure**

A while(true) loop (thus a never ending process) is inferred only if an unconditional feedback path is reflected in the diagram (in Figure 3a, when the a) loop condition is not attached). In the case of Figure 3a the loop do... while (condition1) is inferred. if-else-if and switch statements are generated from the conditional structure in Figure 3b. From each atomic action, a C/C++ function call preserving the
name given in the diagram is generated, where the arguments list corresponds to the set of objects that composed the inputs and the outputs of the action.

A channel access method is inferred from each communication action (AcceptEventActions and SendSignalAction). Figure 4 shows the code translations from the communication actions to the corresponding SystemC channel accesses. In this case, the concurrency resource and the corresponding communication media

![Diagram](image-url)

**Figure 4** Mapping among communication actions and SystemC channel accesses

are in the same Component, thus channel instances within the same module are inferred. Then, the generator prefixes accesses with the channel instance name, taken from the MARTE diagram (e.g., ch.access()). However, in some cases, when the communication media are not included in the same component as the concurrency resource, the communication is established through port links. When this kind of connection occurs, the communication actions are related with port connections in order to access the service provided by a communication media. In this case, the SystemC code that is obtained from the communication action is `port_name->channel_method (variable)`.

As is shown in [1] and [4] this model-based methodology is formally supported. The UML/MARTE-SystemC mapping is formally supported by Formal System Design (ForSyDe). ForSyDe provides the formal foundations in order to maintain the consistency among UML/MARTE models and the SystemC executable specifications. A ForSyDe model provides an abstraction of the semantics associated with the UML/MARTE model (in terms of concurrency structure, time semantics and behaviour) which could be used as a reference model for any model generated from it. In this manner, the SystemC executable specification generated contains the same behaviour semantics as its corresponding UML/MARTE model since both system models have the same ForSyDe abstraction. This implies that the transformation is correct-by-construction.

**V. CODE GENERATOR: FIRST PROTOTYPE**

In order to implement the MARTE-based methodology, a first code generator prototype has been implemented. The code generator has been written as a set of generation templates written in the standard MTL language [6]. This makes the generator open and portable to different generation engines. Moreover, MTL sources unambiguously document and define the generation rules. The development has been done through Acceleo [7], a code generation framework fully integrated in Eclipse.

In this first prototype, the code generator enables the C&C structure to be obtained. In this case, the code generator can identify the communication media that capture the semantics of the KPN and BKPN. Additionally, the code generation provides the internal hierarchy structure composed of Components/modules and the corresponding channel bindings that connect these elements.

The other modelling system aspect that is dealt with by the code generator is transformation of the activity diagrams into the corresponding SystemC code. This generator implementation presents two challenges. The first one is to recognize the different path structures (infinite and loops, conditional statements, etc.) in the activity diagram. This task must be able to identify the different nodes of the diagram (merge node, decision node, etc.) and specify the corresponding functional structure. In this way, the code generator runs the activity diagram, identifying the different path structures and selecting the suitable MTL file. The following code lines show a pseudo MTL language to illustrate how each path structure is distinguished.

```mtl
v1 = channel1.read();
for (int i=0; i<p; i++) {
  v2[i] = channel2.read();
}
function (v1, v2, v3, v4);
channel3.write(v3);
for (int i=0; i<q; i++) {
  channel4.write(v4[i]);
}
```

The other challenge to be dealt with is the translation from the communication actions to the corresponding channel accesses. This generation process has to identify the kind of behaviour semantics captured in the channel and whether the channel access is done directly or by means of a port access.

```mtl
function (v1, v2, v3, v4);
channel3.write(v3);
for (int i=0; i<q; i++) {
  channel4.write(v4[i]);
}
```
The following pseudo MTL code lines distinguish between a direct channel access and a port access:

```
[comment Looking for a communication media owned by the component C/
[for (p : Property | C.ownedAttribute)
  [if p.isStereotypeApplied('CommunicationMedia'))]
  [comment A Communication media with the name of the corresponding communication action/
  [if (p.name.equalsIgnoreCase(channel_name))
    [if p.isInternal()]
      [channel_conection(p)/
    [else]]
      [port_connection(p)/
  [else]]
[/if][/if][/for]
```

```
C. Video Decoder
```

The generator prototype has been verified, by applying it to a video decoder. Specifically, this video decoder is based on the Reconfigurable Video Coding (RVC) decoder architecture [14]. The RVC architecture divides the decoder functionality into a set of functional units (fu) that implement the different functionalities that take part in the video decoding process.

The video decoder is described by UML/MARTE models that capture the video decoder functional unit’s architecture and a description of the functionality implemented by each fu. Then, the code generator prototype produces the SystemC skeleton composed of modules, ports, channels connections and process. Additionally, a behaviour structure of each system process is provided which, when compiled with the pure functionality provided by external C/C++ libraries, produces a SystemC executable. An excerpt of the video decoder model is shown in Figure 5.

Regarding the functionality modelling, the activity diagram in Figure 6 captures the functionality carried out by the MGB fu. The diagram shows the access to the specific channel connected to the MGB fu, the different atomic functions implemented and the different loops and condition structures that define the component behaviour.
A portion of the SystemC code obtained from this diagram is:

```c
#include "MGB.h"

void MGB::mgb_fu_process(void){
  int var_mgb_ConfPar[6];
  ...
  while (true) {
    for (int i=0; i<6;i++) {
      from_framedec_MGBin->read(var_mgb_ConfPar[i]);
    }
    Init_QFS (var_mgb_ConfPar,var_DRC_ConfPar);
    for (int i=0; i<2;i++) {
      to_DCR_out->write(var_DRC_ConfPar[i]);
    }
    do {
      Decode_Type_Block();
      if (Luma_Block) {
        Decode_Luma_Block(var_ac_coeff_DCR);
        to_DCR_out->write(var_ac_coeff_DCR);
      } else if (Chroma_Block) {
        Decode_Chroma_Block(var_ac_coeff_DCR);
        to_DCR_out->write(var_ac_coeff_DCR);
      }
    } while (more_blocks);
  }
}
```

VI. CONCLUSIONS

In this paper, the authors have presented a set of mapping rules in order to obtain SystemC executable specifications. These mapping rules enable the code generation of the C&C structure and the behaviour structure of the functionality associated with each concurrent resource. These mapping rules are formally supported, as was explained in previous papers. Specifically, this paper is focused on the presentation of the mapping rules related with the behaviour structure transformation process.

In accordance with these mapping rules, a first prototype of code generator has been implemented. The development has been done through **Acceleo**. In order to validate this first prototype, it was applied to a video decoder, obtaining the corresponding SystemC executable.

The development of the code generator is ongoing. Future work will cover formalisms such as Communicating Sequential Processes or Synchronous semantics in order to provide more modelling capabilities to enrich the behaviour semantics of the system to be modelled.

VII. REFERENCES


This work has been financed by the European ICT projects SATURN (FP7-216807) and COMPLEX (FP7-247999).
Generation of Abstract IP/XACT Platform Descriptions from UML/MARTE for System-Level Performance Estimation
Towards a MARTE to IP/XACT Generation Framework of HW platform descriptions for a DSE Multi-level Performance Estimation Framework

F. Herrera and E. Villar
Microelectronics Engineering Group, TEISA Dpt.
University of Cantabria
Santander, Spain
{fherrera, evillar}@teisa.unican.es

Abstract—UML/MARTE is enabling the development of methodologies for the specification of a whole real-time system, and of holistic MDA methodologies where the UML/MARTE description is taken as the source for different design activities, such as system-level performance estimation and implementation refinement. A crucial issue to make these methodologies working and efficient is the development of tools, such as code generators, able to handle and produce from the UML/MARTE model the level of information required by each specific design activity. In this line, this paper proposes a framework for the automatic generation of abstract IP/XACT descriptions of the HW platform of a system, synthetic and suitable for system-level performance estimation. This paper is a first step towards a generation framework, able to produce IP/XACT descriptions fitted to the information needs of the different design activities. While this work focuses on high-level performance estimation, the framework is aimed to enable the production of more detailed IP/XACT descriptions able to feed more detailed performance estimation frameworks and implementation tools.

I. INTRODUCTION

Embedded systems are becoming more complex everyday. Their design is becoming more difficult since it requires dealing with a growing amount of information. Reaching a competitive and efficient implementation requires enabling an early description of the whole system, which facilitates its understanding and an early and fast, but enough accurate, assessment of the involvements of the different implementation possibilities. In this line, Model Driven Architecture (MDA) [1] enables the description of a system through different views. For instance, a use case view of the system can provide information for the generation of the necessary test benches for a full system validation. Such use case view does not need to contain information from other views, such as the inner functionality, or the implementation architecture of the system. A proper handling of the abstraction level is also required. For instance, the generation of functional test benches should require handling less information than the generation of HDL test benches. In any case, handling more information than necessary can be cumbersome and counterproductive. In this line, MARTE [2] is a standard UML profile which is enabling the development of different methodologies [3][4][5] for building specifications which capture all the necessary information for the analysis and design of a real-time system. Regarding such analysis, the great variety of platform possibilities and mappings is leveraging the development of fast performance estimation technologies, such as SCoPE [6].

In this context, COMPLEX [7] is a holistic design space exploration (DSE) framework which supports UML/MARTE for the system description. A set of generators extract from the UML/MARTE model the information required to feed performance estimation tools. These tools are, in turn, driven by an exploration tool which decides the next point to be explored in the design space.

In order to make the COMPLEX methodology efficient, easy to use, and scalable, it is necessary that the generators are able to extract from the UML/MARTE specification a synthetic description, only with the information required by the system-level performance estimation tools, and under a format where such information can be grown with the additional information which will be required by further design activities, such as a detailed performance estimation and implementation.

This work has been funded by the COMPLEX (FP7-247999) project.

Figure 1. Generation of the Abstract IP/XACT description in COMPLEX.
Specifically, this paper tackles the extraction (thick arrow in Figure 1) from a COMPLEX UML/MARTE description of the HW platform information required by system-level performance estimation tools. This extraction has several distinctive points. First, the platform information from the UML/MARTE description will be extracted as an IP/XACT description. IP/XACT [8] is a well accepted standard for the exchange of IP components at electronic system and RTL level [9], and already adopted by commercial tools [10]. Second, this paper specifically focuses on the generation of an abstract IP/XACT description. It is abstract of generic, e.g., it uses a bus of reduce set of parameters, instead of an OPB or an AMBA model with a detailed set of characteristic parameters. It is also abstract in the sense of synthetic, because it only contains the information necessary, but sufficient, to serve as input to the COMPLEX system-level performance estimation chain, specifically to SCoPE [6].

Third, the generator and the produced IP/XACT description are scalable. That is, an IP/XACT description with more information suitable for lower-level performance estimation tools (slower but more accurate), or even for implementation, can be generated by adding further information (extracted from the UML/MARTE model) to the abstract IP/XACT description. A separated dealing of this information enables the development of a MARTE to IP/XACT generator suited to a multi-level performance estimation framework.

The rest of the paper is structured as follows. Section II overviews the previous work. It includes an introduction to the type and format of IP/XACT format generated. In section III, the UML/MARTE methodology is introduced, focusing on the description of the HW platform. Then, the generator is explained in section IV, where the more important translations rules implemented and the main features of a generator prototype are explained. Finally, section V provides the main conclusions and future lines of this work.

II. PREVIOUS WORK

A. COMPLEX and SCoPE

In the MARTE COMPLEX flow, there are three main phases/tools. First, the specification methodologies/tools for capturing in UML/MARTE the system description; second, the generation of the SystemC executables, whose execution provides performance figures; and, third, the DSE tool, able to select the next points of the design space to be explored and thus drive the exploration. In COMPLEX, the generation of the SystemC executable is actually split into two phases. In a former step, a set of generators is in charge of producing all the input files which are required for the generation of the SystemC executable. These files do not contain SystemC code, but the information which, in turn, is used for the generation of the SystemC executables.

Figure 2 sketches the three main formats automatically generated in the COMPLEX generation flow. A COMPLEX specific format (CFAM) serves for the description of the system application, including its structural information, its functionality, and the inherent concurrency. The CFAM will also include the real-time features included in the MARTE description. From the CFAM description, a SystemC platform independent model (PIM) will be generated, which enables a functional validation of the system, and moreover, feeds further stages of the COMPLEX flow.

![Figure 2. SystemC executables are generated in two steps.](image)

The automatic generation from MARTE in COMPLEX will also produce XML files which contain information relative to the description of the platform, and to the allocation of application components into platform components. This XML information, which can be split into different files, will also reflect all the configurable parameters which are susceptible for exploration, and which can be tuned by the DSE tool. This highly configurable XML is a suitable format to enabling the generation of configurable platform specific models in SystemC.

Specifically, SCoPE [15][16][17] is one of the performance estimation frameworks involved in COMPLEX. In COMPLEX, SCoPE will be in charge of system-level performance estimation. This means that SCoPE will be able to enable the faster iterations in the DSE cycle. For it, SCoPE relies on native simulation [16][17], and requires different information of the system configured, which regards to the application, to the platform and to the way the application components are allocated to the platform components. With abstract information and an abstract model of the system, SCoPE is able to provide different metrics of the system, such as power consumption, time latencies, usages of CPUs, etc.

From its origin, SCoPE has been able to support as input a concurrent application relying on a RTOS API (currently POSIX, WIN32 or uC/OS-II). In COMPLEX, SCoPE is being extended to SCoPE+, in order to support the higher level CFAM API. As SCoPE, SCoPE+ will be able to consider the performance effects of custom hardware, encrusted as TLM SystemC models, directly plugged to the system bus. Moreover, SCoPE already supports as an XML input the description of the platform, of the allocation of SW components to HW components, and the configurability of the platform. Finally, in a previous work [19], SCoPE was enabled to admit the description of the HW platform as an IP/XACT description, thus separated from the rest non-standard XML information.

In COMPLEX, the generation of an IP/XACT description of the HW platform is foreseen. Such an IP/XACT description reflects a fixed architecture (the optimum one, found by the DSE phase), and contains all the
information required for assembling the HW platform, provided the IP/XACT descriptions of each platform component is available. It enables the later generation of a detailed virtual platform (which enables the validation of the exploration results, with more accurate performance figures) and as source for the implementation flow.

In COMPLEX, the generation of the configurable a PSM SystemC executable requires from SCoPE+ to be able to read the application description as CFAM code, and the platform description, allocation information, etc, as XML code. Beyond the COMPLEX objectives, provided that SCoPE+ is able to admit an abstract IP/XACT as input for the description of the HW platform, this paper proposes enabling the direct generation of such synthetic IP/XACT descriptions already in the phase where generator are producing code for feeding the system-level DSE. Once the generation phase has been completed, the generation of the IP/XACT description of the platform should be seen as a grown version of the IP/XACT code employed for the DSE phase.

B. IP/XACT description of the HW platform for High-Level Performance Estimation Models

In [19], a plug-in which enables that SCoPE read the HW platform information as an IP/XACT description, while the rest of the platform (SW and application components, and mappings) is described in a specific XML format suitable for DSE. Different files are used for this specific XML description and for the IP/XACT description.

The IP-XACT hardware platform description can consist in general of several files. Each of them must contain as a root element one from the top seven element definitions. At least one file must contain the design element definition as the root element:

```xml
<spirit:design
xmlns:spirit="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.4"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.4 ">

<spirit:vendor>teisa</spirit:vendor>
<spirit:library>scope</spirit:library>
<spirit:name>miplatform</spirit:name>
<spirit:version>1.0</spirit:version>

<spirit:componentInstances>
  ...
</spirit:componentInstances>
<spirit:interconnections>
  ...
</spirit:interconnections>
</spirit:design>
```

This root element is equivalent to the top schematic representation of the hardware platform, and is identified by means of its vendor-library-name-version identifier, that is, its VLNV identifier [8].

Hanging from the root design element, there are two main sections. The entry <spirit:componentInstances> delimits a section which contains component instances, recognized by the <spirit:componentInstance> IP/XACT entry. A second section, defined by the entry <spirit:interconnections> is in charge of containing the set of interconnections among the aforementioned component instances.

In the concise IP/XACT representation required by SCoPE, the component instance only requires a basic information defined by three attributes, which includes the component instance name (<spirit:instanceName> entry) and the reference to the component (<spirit:componentRef> entry). This component reference is done through the VLNV identifier of the component. The third entry (<spirit:vendorExtensions>) is used to facilitate the automatic generation of high-level performance estimation and simulation models.

```xml
<spirit:componentInstance>
  <spirit:instanceName>i_bus</spirit:instanceName>
  <spirit:componentRef>
    spirit:vendor="teisa"
    spirit:library="scope"
    spirit:name="bus"
    spirit:version="1.0"
  </spirit:componentRef>
</spirit:componentInstance>
```

In effect, the Spirit standard is focused on the capture of structural information. The VLNV identifier serves to give a link to an IP/XACT component description which, in turn, gives again a structural description. However it does not provided a specific functional semantics to the component. In order to cover this lack, the IP/XACT format read by SCoPE uses context labels formally defined by the SPRINT project in the SCIPiV document [20]. This way, SCIPiV tags are used to identify the component type, and thus facilitate the generation of generic abstract models. SCIPiV tags can be also in component instances. This is the case of the example shown, where the i_bus component instance has a context label <spirit:isNetworkComponent>.

SCPiV tags are used for the integration of generic components in the SCoPE simulation model. SCoPE provides a default support or portfolio of generic components (processor, bus, memory, DMA, bridge, etc). If the Spirit design file contains component instances whose VLNV identifier does not fit with any component description file, they will be considered as generic if possible. Context clauses are used in this case for using the generic SCoPE components. This enables a fast and easy building of a platform specific executable model.
As well as the portfolio of components offered by SCoPE, the SCoPE user can provide and attach to the platform new components such as custom IPs and I/O devices. Moreover, the user can even override the use of generic platform components and use their own performance models. For it, a SystemC model with a TLM interface which can include behaviour and performance annotations has to be provided. In SCoPE, this model can be directly encrusted at a SystemC level. Moreover, the IP/XACT support for the integration of these new components enables making an IP/XACT component declaration (by means of the entry <spirit:component>) separated in a file different from the one which contains the top design (<spirit:design> entry) and the architectural description defined by the component instances and interconnections sections. This enables a modular and cleanly extensible description of the hardware platform. The component description is identified again by a VLNV identifier, which can be referenced by a component instance. The component declaration also includes additional information which enables a connection to the SCoPE platform structure, namely, a section for the bus interface description (<spirit:BusInterfaces>), a section for describing the mappings of the device to the memory map (<spirit:memoryMaps>), and, finally, a vendors extension section with the a SCIPIV tags to specify the type of component.

If a component instance references a component description which does not reference a behavioral model, and it is possible to infer a generic component, then the hardware platform model cannot be generated.

The simple interconnection scheme required by SCoPE for building a fast executable performance model is reflected in requiring a simple description for the interconnections section. Specifically, SCoPE only requires the description of point-to-point interconnections among HW platform component instances. This is done by using the <spirit:interconnection> element for each interconnection. The interconnection is described by its name (<spirit:name>) and by the two connection extremes (using the <spirit:activeInterface> element). The tied component instances are defined by assigning the component instance name to the spirit:componentRef attribute of each spirit:activeInterface. In the following example, the component instances “i_mem” and “i_bus” are interconnected. Additionally, the IP/XACT description reflects the master or slave character of the component attached (e.g., the “i_mem” has a slave role). Connections of the type master to bus, and slave to bus (as the one in the example) can be described. This serves to reflect the simple computational scheme of a high-level simulation tool such as SCoPE, where a key for fast simulation is the typical master-slave configuration, where slaves are reduced to functional calls, and the number of masters is kept as reduced as possible.

```
<spirit:interconnection>
  <spirit:name>c2</spirit:name>
  <spirit:activeInterface>
    <spirit:busRef="Slave">
      <spirit:componentRef="i_mem" />
      <spirit:componentRef="i_bus" />
    </spirit:busRef>
  </spirit:activeInterface>
</spirit:interconnection>
```

Another IP/XACT capability supported by SCoPE is the definition of configurable platforms. Through the entry <spirit:configurableElementValue>, a value of a component attribute can be fixed or even left as a parameter (_MEM_ADDR_) of the generated executable specification. Then, this parameter needs to be passed to the generated executable specification before the execution of executable specification.

III. UML/MARTE SPECIFICATION METHODOLOGY

This work is in the context of the COMPLEX code generation framework, which takes as input a UML/MARTE system description under the UML/MARTE methodology. A detailed description of this methodology is out of the scope of this paper. Here a brief introduction of the overall methodology is given, to later focus on all the information which regards to the description of the HW platform.

The COMPLEX UML/MARTE specification methodology is a component-based approach which enables the description of the whole system, including the application, the platform, and the allocation of application components into hardware components. Moreover, a COMPLEX UML/MARTE specification enables the capture of a set of real-time features associated to the application components, and the definition of an environment, clearly separated from the system description which defines the different use cases and their related input stimuli.

A relevant feature of the COMPLEX UML/MARTE specification methodology is the separation of concerns. Specifically, this means that the different information related to the system description is separated into different views: namely, (1) a data view; (2) a functional view; (3) a concurrency and communication (C&C) view; a platform view (4); and architectural view (5); and (6) a verification view. The former five views are devoted to the system description. Among them, the platform view and the architectural view are the ones which contain information for the extraction of the HW platform architecture, and thus for its IP/XACT description. Each view is captured as a UML package with a COMPLEX specific stereotype. Specifically, the <<PlatformView>>, <<ArchitecturalView>> stereotypes.

The platform view declares both, the SW components (RTOS, drivers) and the HW components (processors, memories, DMAs, buses, etc) which will build up the platform. It is done by including in the <<PlatformView>> package UML components with different MARTE stereotypes. Specifically, for HW components, stereotypes from the MARTE HRM subprofile <<HwHRM>>, <<HwBus>>, <<HwBridge>>, etc are used. The main software components are RTOS, which are captured as UML components stereotyped with the MARTE GRM <<Scheduler>> stereotype.
The architectural view contains a single component (a UML component stereotyped with the <<system>> COMPLEX stereotype) which reflects the internal architecture of the system by means of a UML composite diagram. This diagram contains:

- the application component instances (captured as parts typed as any of the components of the C&C view (stereotyped as <<RtUnit>>) and their interactions. (connectors between part ports).
- the platform (software and hardware) component instances (captured as parts typed as any of the components of the platform view) and the interconnections of HW component instances (captured as port connectors).
- the allocation of application component instances to platform component instances, such as RTOS instances and instances of computation resources, such as processors, or custom hardware. It is captured as associations with the <<allocate>> stereotype.
- information which distinguishes the methodology, such as DSE parameters and rules. They are specified as comments stereotyped with COMPLEX specific stereotypes such as <<dseScalarParameter>> and <<dseRule>>, which enable to define as tunable certain attributes of component instances.

Further information in the platform and architectural views corresponds to the values of the attributes associated to the different instances. The attribute values in the platform components are understood as default values inherited by any instance. An association to a <<dseScalarParameter>> comment can change the value of an attribute for a given instance. Among the different tunable attributes, restMult is a is used to specify in a compact way the number of instances associated to a single part in the UML composite diagram of the system architecture. Other attributes state non-functional properties, e.g., the frequency property for a HWProcessor instance, of NFP Frequency UML/MARTE type.

The COMPLEX UML/MARTE specification methodology has been put into practice over a toolset based on MDT Papyrus [11], which is integrated in Eclipse Helios [12]. Despite this toolset does not provide specific features for the separation of views, the COMPLEX UML/MARTE methodology enables by means of the mentioned stereotyped packages. By keeping the architectural information (instances and connectors) and configuration information in a single view (architectural view), the need of any coherence check among views is eliminated.

IV. GENERATOR

A. Transformation Rules

Although the platform and architectural views contain all the information necessary for the generation of the IP/XACT HW platform description required by SCoPE, not all that information is required for the generation.

Figure 3 provides a basic view of the MARTE to IP/XACT mapping implemented by the generator.

The generator will produce a single file with the spirit:design entry. The generator does not produce files with spirit:component entries, since these are assumed to be available. However, the components are declared within the MARTE specification. Therefore, the generator basically produces the architectural description of the HW platform.

![Figure 3: Basic sketch of the MARTE to IP/XACT mapping.](image-url)
interface. In order to produce the `spirit:busRef` entries, the generator needs a deeper navigation to determine the type of components tied and if it is master or a slave connection.

Finally, the generator infers from the `dseScalarParameters` comments, the `spirit:configurableElementValue` entries.

**B. Generator Prototype**

A prototype of the MARTE to IP/XACT generator has been implemented. The generator has been written as a set of generation templates under the standard Model to Text Language (MTL) [13]. This makes the generator open, easily extensible and portable to different code generation engines supporting MTL (e.g. Acceleo, Xpand, Jet). Specifically, the generator has been developed with Acceleo MTL [14], a code generation framework fully integrated in Eclipse Helios [12].

The generator is defined as a MTL module, the UML2 metamodel as input:

```
[module mart2ipxact('http://www.eclipse.org/uml2/3.0.0/UML')/]
```

This environment has facilitated the modular programming of the generator, through a hierarchical set of templates and navigation queries using OCL language as a complement to MTL. Notice that the COMPLEX MARTE2IPXACT generator requires a recurrent navigation of the specification from the architectural view to the platform view, and along the different elements which gets more complex in certain cases. For instance, to recognize if a certain UML connector relates HW components, the navigation goes from the end point of the connector through a port, its owner part, and then its component type. Then it checks that such component type belongs to a set of HW components, determined through a query.

The recognition of the application of MARTE stereotypes in the model is easily done through the following MTL code structure:

```
isStereotypeApplied(getApplicableStereotype('MARTE:…'));
```

The generator implements a set of basic checks, while the specification is navigated. For instance, the generation checks that there exists the two required views (platform and architecture), that the architectural view is not empty and that contains a single system component, that there is a minimum set of HW component instances in the architectural view, etc.

Finally, the generator also includes a header, indicating the automatic generation, and XML comments to let track the UML/MARTE elements detected in the specification which have led to the associated IP/XACT code.

**V. CONCLUSIONS**

In this paper the generation of generic and concise IP/XACT descriptions from UML/MARTE system descriptions suitable for DSE of real-time complex embedded systems is addressed. Specifically, this generation reads and traverses the platform and architectural views of a COMPLEX UML/MARTE specification and produces a file with the architecture of the hardware platform in IP/XACT format. This IP/XACT code generated contains a minimum set of elements plus some functional information (SCIPv tags) which synthetically describes the hardware platform and is directly readable by SCoPE for the automatic generation of an executable for high-level functional and performance estimation.

Future work will cover the generation of more detailed IP/XACT descriptions, enabling the generation of more detailed, and thus accurate, performance models, and moreover the assembly of platform components for actual implementation. Such work will take as reference the IP/XACT descriptions admitted by a Magillem generation tool. This tool encourages generators able to produce the aforementioned detailed SystemC virtual platforms.

**REFERENCES**

[17] [www.teisa.unican.es/scope].
Combining SystemC, IP-XACT and UML/MARTE in model-based SoC design

Jean-François Le Tallec∗†, Julien DeAntoni∗†, Robert de Simone†, Benoît Ferrero†, Frédéric Mallet∗† and Laurent Maillet-Contoz†

∗UNS, Nice Sophia-Antipolis, France †AOSTE Team, INRIA Rocquencourt / INRIA Sophia Antipolis / Laboratoire I3S, France ‡STMicroelectronics, Crolles, France

Abstract—Modern SoC design may rely on models, or on high-level description languages. Although very close, the benefits obtained from either sides can be substantially different (and mismatch may occur). The IP-Xact formalism, now a standard (IEEE 1685), was introduced to help assemble component IP from distinct sources into an integrated design. Components could be expressed in high-level HDLs such as SystemC, so should be the full design after translation. Experience shows that in fact this is hardly the case, specially in publicly available methods and tools. The present contribution goes one step further, into linking SystemC designs to their IP-Xact structural representation by translation. It then exports the resulting IP-Xact model into the UML/MARTE profile modeling framework, to allow annotating existing models with additional information (again in a publicly available fashion, as opposed to vendor extensions). Even if our approach is still far from being complete, it bridges a number of gaps induced by the combined use of SystemC and IP-Xact.

I. INTRODUCTION

Design of digital circuits was always involved with many representation formalisms. Some of them are formal or engineering models of such circuits, some are programming languages initially aimed at simulating such models. Combinatorial and sequential netlists, Mealy and Moore FSMs, process networks fall into the first range; Verilog, VHDL and SystemC language in the second. In particular, SystemC is becoming a de-facto standard for high level description languages. Although very close, the benefits obtained from either sides can be substantially different (and mismatch may occur). The IP-Xact formalism, now a standard (IEEE 1685), was introduced to help assemble component IP from distinct sources into an integrated design. Components could be expressed in high-level HDLs such as SystemC, so should be the full design after translation. Experience shows that in fact this is hardly the case, specially in publicly available methods and tools. The present contribution goes one step further, into linking SystemC designs to their IP-Xact structural representation by translation. It then exports the resulting IP-Xact model into the UML/MARTE profile modeling framework, to allow annotating existing models with additional information (again in a publicly available fashion, as opposed to vendor extensions). Even if our approach is still far from being complete, it bridges a number of gaps induced by the combined use of SystemC and IP-Xact.

While modeling and programming formalisms do complement one another, they do not always coincide as well as one should hope. This is certainly so because they follow distinct goals. Analysis models aim at faithfully representing physical objects, while allowing abstractions from details, or considering additional relevant views (consumption, timing closure). Programming languages promote execution efficiency to allow simulation of very large circuits, sometimes at the cost of modeling accuracy. The interplay of models and programming languages has thus always been a big issue for correct design (as for instance, in synthesizability requirements). The matter has been renewed with the advent of model-driven engineering techniques (based on UML or similar formalisms), which should provide a middle point between the demands of formal models and mathematical properties on the one hand, programing and design efficiency on the other one. Currently the IP-Xact standard initiative (initiated in the Spirit consortium and now handled at Accelera) aims at providing a dedicated ADL (Architecture Description Language) to model the hierarchical structure of SoCs and interconnects, and support representation of additional feature aspects in a normalized way. Ties between IP-Xact and SystemC should be obvious, but currently they remain rather obscure and implicit, hidden in proprietary tools and implementations.

Our present work consists of two parts. First, we capitalize of previous efforts by others to extract a complete structural model in IP-Xact from a SystemC program during its so-called elaboration phase; it provides a way to compose and assemble SystemC programs more easily and modularly. Second, we consider the representation of these IP-Xact structural descriptions into the MARTE profile of UML. MARTE is the UML extension meant to deal with Real-Time Embedded aspects of systems, which suits closely digital circuits. MARTE allows also to represent various non-functional aspects (timing, consumption,...), in a standardized fashion. This comes as an extension to IP-Xact feature for introducing extra information, usually restricted to proprietary Vendor Extensions. In MARTE, additional features will become integral public parts of the model. We choose to take test cases in external library to test the robustness of our tool confronted to different coding styles. For that purpose, we stress our tools on the SoCLib library and generic SystemC examples.

The benefits of our results are two-fold. To the best of our knowledge no explicit translation from SystemC to IP-Xact that would preserve and even identify and promote the hierarchical structure was available to this date. Also, the potential ability of a dedicated format such as IP-Xact to support annotations for representation of additional feature aspects was again not something available at this stage. Our contributions should help in the future experiment with various such additions, with the main impact of using the same basic skeleton model to add or extract different feature aspects that can efficiently be dealt with for mathematical analysis by a large spectrum of existing tools. In this paper, we will present in section II different existing technologies on which our approach/tool is relying on. Then, we will explain more precisely in section III-A how we built it and what possibilities it permits. We will conclude in section IV by enumerating earthly perspectives for the future works.

This paper has been partially supported by the French ANR project HELP (ANR-09-SEGI-0006).
II. Overview

A. SystemC

SystemC\(^1\) is a Hardware Description Language, meant to represent circuits and SoCs at various levels of abstraction (in particular at RTL and TLM levels) [5]. Conceived as library extension of C++, it benefits from data types and compiler environments from this host language. It adds provisions for parallel threads, signal wires, clocks, timing features, low and high level communication mechanisms. These extensions are semantically dealt with by a specific non-preemptive scheduling discipline, for simulation of designs. Simulation goes through two successive phases. First, the elaboration phase instantiates the parallel processes and the static network of components (i.e., the sc_modules and their interconnects), as requested by the program in an initial object creation part. Then the actual behavioral simulation itself takes place, combining the individual component bodies according to the scheduler. Simulation itself can be untimed (causal), cycle-accurate, approximate or even loosely timed. SystemC designs are usually strongly influenced by formal modeling with so-called Models of Computation and Communication (MoCC). It can be traced back to the pioneering work of Daniel Gajski and fellow co-authors on SpecC [4]. Also the seminal SystemC reference [5] mentions Kahn Process Networks. Modeling Kahn Process Networks, CSP and Synchronous Reactive systems in SystemC is also explored in [6]. Heterogeneous modeling in SystemC is addressed in [12], [17]. However, the connection remains implicit, or even unclear, when looking at actual programs. The elaboration phase can be seen as the construction of the structural parts of models (process networks and interconnect topology), while component behaviors can usually be seen as some form of interacting FSMs (possibly with timing). One of our goal is to extract such modeling information in an explicit way (in IP-Xact for structure and interfaces, in later work in UML MARTE for component behaviors).

B. IP-XACT

IP-Xact\(^2\) is an XML format dedicated to the design, integration and reuse of IP (Intellectual Property) components from various vendors into larger designs by enabling automatic configuration and integration through industrial tools. IP-Xact was created by the Spirit Consortium and is now handled by Accelera, recently becoming the IEEE 1685 standard. IP-Xact provides a means to describe information relative to the structural part of a design. A component definition describes the interface of a component and contains a reference to a file that describes the implementation of its functional behavior in a specific HDL. A design is a set of component instances and links that interconnect these instances.

Because correctness is an essential concern for integration and reuse, functional and extra-functional properties can be embedded in a component definition or a design by using so called vendor’s extensions. The format and the kind of information contained in a vendor extension is not part of the standard, and thus can not be shared across different vendors tools. This prevents integration of IP developed individually in different tools. Another limitation is the lack of mechanism to easily handle parameterizable structures. These are the main reasons why we believe that more generic definition formalisms such as those found in UML could be useful here.

C. UML/MARTE

MARTE\(^3\) [11] is the profile extension to the UML modeling framework aimed to deal with Modeling and Analysis of Real-Time Embedded systems. It inherits from UML several diagrammatic styles (such as components for structural design, hierarchical FSMs and data-flow activity blocks for behaviors, amongst several others). It also provides standard annotation features (there called “stereotypes”) to represent functional and extra-functional properties, as well as a way for the user to introduce others. As a result, further specialization of MARTE could allow encoding IP-Xact notions, as well as easy extensions like the IP-Xact vendor’s extensions, but this time publicly exposed. The emphasis put in model-driven engineering (as around UML) on model transformation should then allow the relevant information to be directed to whichever proprietary tool may understand how to deal with them. Furthermore, it embeds a logical time model [2] to allow description of constraints to formally link different views of a model such as consumption, timing or safety and possibly a TLM to RTL refinement way [7]. Repetitive structure modeling (RSM) package defined in the standard UML MARTE profile also simplifies the representation of complex parameterizable structure.

D. Our approach

![Figure 1. Common SoC Design](image)

Classically, SoC platforms are depicted in a block diagram fashion as in figure 1. But such an informal picture is not a true model, as it lacks usually important information. As the purpose of IP-Xact is to promote platform assembly using IP blocks, it requires further information of the port interface and interconnect features. Furthermore, MARTE profile could provide graphical editing and formal support for annotation in

\(^1\)The Open SystemC Initiative [www.systemc.org](http://www.systemc.org)

\(^2\)The Spirit Consortium [www.spiritconsortium.org](http://www.spiritconsortium.org)

\(^3\)UML Profile for MARTE [www.omgmarote.org](http://www.omgmarote.org)
distinct non-functional domains (e.g., power domain). It is our main goal to model such a platform as in figure 1, but with possibly all the details needed to understand it fully (such as the protocol description).

Thus, our global vision is that composite SoC design descriptions integrating IPs from different sources should be provided in the IP-Xact fashion, certainly extended with useful annotation features in a way less obscure than “vendor extensions”. Generic model-driven engineering, in our case embodied in the UML/MARTE profiles, could help prototyping such extensions. In turn, SystemC could be used as an efficient target simulation language, with programs benefiting from “correct-by-construction” features obtained by early model analysis. Individual component behaviors could also be provided as SystemC descriptions (or with formal MoCCs).

Still, there are currently many more design descriptions directly provided in SystemC, and lifting them back up into model-level descriptions in IP-Xact could greatly help populate the design methodology. This is why a model extraction scheme from SystemC programs is so desirable. It is the main topic of this paper, pointing out the potential difficulties as well as openings of the approach.

![Figure 2. Global Tool Flow](image)

So, we consider the automated bottom-up translation from SystemC to UML/MARTE, through IP-Xact, as shown in figure 2. To translate from SystemC to IP-Xact, we extract information from the original program using two different methods. The first method consists in running a simulation along its elaboration phase, and recover from that a model for the fully instantiated network of components; this part is directly indebted to former work on PinaVM (see below). The second method uses static analysis introspection of the SystemC code to recover structural syntactic information that were not preserved during the compilation process; it relies heavily on the Doxygen C++ static analyzer (again described below). After these two were conducted in parallel, we merge their results into a complete IP-Xact view that conforms to the IEEE 1685 standard description. We defined and realized a second translation, this time from IP-Xact to a corresponding subset of UML/MARTE. It uses advanced model transformation techniques. The main point here is that resulting models could then easily be extended to allow supplementary annotation features (such as timing or low-power), in a public representation format (as opposed to current vendor extensions in IP-Xact). While this part is still underdeveloped, we can envision potential further work, for instance in describing early abstract designs that are “bus protocol agnostic”, and can then be refined to accommodate AMBA or OCP-IP requirements. Corresponding vendor extensions could then automatically be synthesized in a way compliant to IP-Xact. We have already implemented a reverse translation from UML/MARTE to IP-Xact, currently bare regarding such extended annotations, but which could easily support them once defined.

E. Inspirational works

As mentioned above, the translation from SystemC to IP-Xact requires two distinct methods to provide results, one based on run-time simulation of the elaboration phase, one based on static analysis of the source code. We now describe in turn the two already implemented methods, due to others, upon which we built our process by proper modifications and enhancements. One main highlight of our approach in SCiPX is to combine both techniques to get the final combined models.

a) Elaboration run-time: PinaVM: PinaVM [9], [10] is a SystemC front-End based on the LLVM compiler. It provides an abstract representation of a SystemC programs after the elaboration phase (i.e., it provides an abstract view of the network of components). It also provides an extendable project structure to plug specific backend, allowing the manipulation of this representation. The abstract representation provided by PinaVM is well suitable for verification / validation of the behavioral part but, due to the use of precompiled information, lacks of static information. For instance, as for every C++ programs, the precompilation remove information on the attribute names and mangle information on attribute types. Moreover, PinaVM does not focus on architectural concern and does not directly allow the translation of the representation into a model view (either IP-Xact or UML-based). We chose to take advantage of the already existing project structure and to develop a back-end that have additional information from static SystemC code analysis.

b) Static analysis: Doxygen: Doxygen is originally a documentation generator but can also be used as a static code source analyzer. It permits us to retrieve information about the component definition. The goal is to retrieve information that lacks in PinaVM like the name of the attributes, etc. Of course, Doxygen can not give any relevant information about the elaboration phase or the component network. It appears that PinaVM and Doxygen can fit together and provide enough information for the construction of both IP-Xact component definition and IP-Xact design. From these IP-Xact description, it is then possible to make an import into UML/MARTE. These steps are detailed in the next sections.

F. Related Works

The connection of SystemC to more engineering models and meta-modeling frameworks (such as UML) is also not entirely new [15], [3]. Early connections between IP-Xact and

\*Doxygen www.doxygen.org
UML models were presented in [14], [16], [8]. The present work is original in that it provides a general translation of the structural aspects of SystemC programs into engineering models, and allows this translation to be later completed with various annotation aspects that capture more views of the design (like behavioral aspects for instance). While we focus on the code to model transformation at this point, to benefit from existing collections of SystemC programs, we are eager to reverse transformations so as to “synthesize” SystemC programs from high-level models (something which would sound obvious from many claims in the literature, but is not yet achieved by any means to the best of our knowledge).

III. Detailed Tool Description

A. SCiPX description

We now explain why our tool requires combination of dynamic execution (elaboration phase in PinaVM) together with static analysis (Doxygen). We illustrate this on the simple design depicted in Figure 3. \( c1, c2 \) are two components of the design, respectively instances of \( M1 \) and \( M2 \). \( p1, p2, p1', p2' \) are ports of the same type \( T1 \). \( c1 \) contains \( p1 \) and \( p2 \); \( c2 \) contains \( p1' \) and \( p2' \). \( p1 \) is connected to \( p1' \) through \( channel1 \) \( p2 \) is connected to \( p2' \) through \( channel2 \). While this information alone allows to create a design view, more information about the admissible number of port \( M1 \) and \( M2 \) instances can accept should also be provided (not always “2” on all instances). The dynamic runtime analysis recovers the involved instances and their port instances as well as the link between the ports. However, they loose information like port naming, exact type of the definition (two ports or an array of size two makes no differences), and other attributes of the component. In the previous example, all ports have the same type and are, after pre compilation identified by their memory address. It is then impossible to deduce which address corresponds to which name. More generally, the name loss makes impossible to differentiate component ports of a same type. The static approach on the other hand, can fill up attribute name information. But, in case of dynamic (and / or conditional) instantiation, is not able to retrieve the component network. So we have to combine static and dynamic approaches in a proper way.

First, retrieve static knowledge about the SystemC component definition; second, retrieve information about component network resulting from the elaboration phase (i.e., need to retrieve run time information); finally merge all these pieces of information in order to have enough knowledge to create an IP-Xact system from the SystemC code.

This process is depicted in figure 4. First, we run Doxygen over a SystemC source code where macro has been replaced in order to get native C++ source code (link n°1). It results in an xml file that contains all class definitions together with inheritance and containment.

This xml file can then be analysed to extract the various component definition (i.e., component, interface, attributes, etc) (link n°2). From the same xml file, a new main program is produced. It does not modify the existing SystemC \( sc\_main \) code. The goal of this generated \( sc\_main \) is to collect offset of each attribute of each components identified by Doxygen, using the standard ANSI C macro \( offsetof() \). Technically, the offset represents the difference between the attribute address and the address of its class. A new xml file result of this. It represents the mapping table between an attribute name and its offset (link n°3). It is important to notice that this mapping depends on compilers/compiling options. Consequently, this code must be compiled with exactly the same options and compiler than the one used by PinaVM and on the same execution platform.

In the next step, the original SystemC program is executed until the end of elaboration by PinaVM. When this point is reached, PinaVM returns to the back-end plug-in an access to its component network internal representation. Now, using the available attribute offset mapping table built before, information about attribute names (and furthermore port names) can be asserted by comparing the address of a port, the address of the owner component and the offset(link n°4).

In order to retrieve the attribute types in a human readable form, we used RTTI (Run-time type information), a way to keep information about an object’s data type in memory at runtime. This has to be done at runtime to make sure types are retrieved even when using template components. It becomes then possible to build a complete IP-Xact design linked to the appropriate component definitions (link n°5).

![Figure 3. Simple example of two communicating SystemC sc_modules](image-url)

![Figure 4. SystemC to IP-XACT details](image-url)

The following source code illustrates a number of declarations that our SCiPX\(^5\) tool is able to handle. The first

\(^5\)AOSTE www-sop.inria.fr/aoste
piece of code is an implementation of a Source component. Assume that there is the mirror implementation for the Target component.

As highlighted on line 2,3 and 4 we deal with dynamic declaration of ports as well as with dynamic port array. As SystemC do not allow instantiation when the elaboration phases is done, any dynamic ports must have been created. Knowing the address of these port objects and the reference of port pointers in a component, a simple comparison between port address and port pointer value permits us to map dynamic ports. Dynamic port array is very similar to dynamic port. The whole array is declared at the same time. The first port of the array is detected as a simple dynamic port. A simple comparison between unmapped port addresses and the size of the port permits us to link unmapped ports to the first port of it array.

```
1 SC_MODULE(Source){
2   sc_out<statType> statOUT;
3   sc_out<dynType> * dynOUT;
4   sc_out<arrayType> * arrayOUT;
5   SC_HAS_PROCESS(Source);
6   Source(sc_module_name name, unsigned arraySize) {
7     dynOUT = new sc_out<dynType>();
8     arrayOUT = new sc_out<arrayType>[arraySize];
9   }
10 }
```

The last piece of code is the instantiation of both Source and Target into an sc_main. Such a declaration is valid as it is simple C++. But, it could not be synthesized by classical static analysis (i.e., without symbolic execution) because it contains loops and conditional statements. Combining the runtime and static approach permits us to overcome this limitation.

```
1 int sc_main(int argc, char **argv){
2   unsigned arraySize=2;
3   bool invertArray=true;
4   Source sourceInst("sourceInst.", arraySize);
5   Target targetInst("targetInst.", arraySize);
6   sc_signal<statType> statSIG;
7   sc_signal<dynType> dynSIG;
8   sc_signal<arrayType> * arraySIG;
9   arraySIG = new sc_signal<arrayType>[arraySize];
10   sourceInst.statOUT.bind(statSIG);
11   sourceInst.dynOUT->bind(dynSIG);
12   for(unsigned i=0;i<arraySize;i++)
13     sourceInst.arrayOUT[i].bind(arraySIG[i]);
14   targetInst.statIN.bind(statSIG);
15   targetInst.dynIN->bind(dynSIG);
16   if(!invertArray) for(unsigned i=0;i<arraySize;i++)
17     targetInst.arrayIN[i].bind(arraySIG[i]);
18   else for(unsigned i=0;i<arraySize;i++)
19     targetInst.arrayIN[i].bind(arraySIG[arraySize−i−1]);
20   sc_start();
21   return 0;
22 }
```

We ran SCiPX on a number of examples found in the standard SystemC library (ver 2.2), and were able to correctly generate IP-Xact representation for most. Examples that did not pass the processing all contained the same feature: to specify connections, these examples do not use sc_port_base which is the information handled in SystemC internals. Thus, asking SystemC internals do not reveal these kind of connections as it is not saved during the elaboration phase. One can argue that the problem comes from a poor SystemC coding style on these examples, but we are considering a work-around this.

B. Ipact2Marte and Marte2Ipact

IP-Xact allows the specification of additional, NFP (Non-Functional Property) annotations only through so-called vendor extenstions, which remain private and not standardized. We used the UML standard profile MARTE as a substitute, where such NFP annotations can be defined and charaterized in an open public way and then possibly translated to IP-Xact in any vendor format that is made available.

Another advantage of MARTE is the possibility to use already existing tools to realize / visualize / edit an Ip-Xact design. For visualization purpose, in addition of the Ip-Xact to MARTE transformation, we developed an automatic di2 generator which permits us to display the extracted design in Papyrus, a free and open source UML / MARTE environment. Another side effect is that behavioral and NFPs can be analysed by different available tools directly at the model level. For instance, works around data flow network allow the computation of static periodical schedule and results can be injected back in the MARTE model.

The transformation from IP-Xact to MARTE is a reverse implementation of the mapping described in [1]. Together with this previous work, we provide a mean to travel between IP-Xact and MARTE. It is then possible to import component definitions and (possibly) a design, to modify (or create) the design in a UML editor, to annotate the design, to analyze it and then to re-generate IP-Xact representation of our extended design keeping retro compatibility with IP-Xact tools. For now, we only prototype a specific vendor extension transformation.

---

5 Papyrus www.papyrusuml.com
from and to IP-Xact. Because vendor extension is a very general extension mechanism, there is no way to provide a generic transformation for that purpose. However, doing it on various vendor extensions promotes them to a first order concern, publicly exposed while keeping the advantage of the algorithm / tool in charge of the extension.

**Figure 6.** MARTE diagram of the example of section III-A

After processing the source code of the example of the section III-A, we have the MARTE model and diagram whose a screen-shot is given in figure 6. To stress our implementation, we ran our tool on the SoC Lib SystemC library. We succeeded in generating IP-Xact component definitions and transforming them into UML/MARTE components as well. Then, we were able to graphically create various designs in Papyrus MARTE with these extracted components. Then, we were able to generate IP-Xact design file from MARTE. As SCiPX, tools to achieve these transformations are also available on AOSTE webpage.

**IV. CONCLUSION AND FUTURE WORK**

An ideal SoC design flow would allow to assemble well understood components models, through well defined structural interfaces. Early analysis at model level could help guarantee correctness properties, in the functional and non-functional domains. IP-Xact goes in this direction by providing clear interfacing and configuration mechanisms, but extra information is currently provided as vendor extensions outside the range of the standard. Also, component models are deferred elsewhere, and in fact provided as HDL code. SystemC ambitions to allow the representation of MoCCs at several levels, but as it relies on code there is no guarantee that programs match this need (even though the language itself is expressive enough for that of course). Since there are currently many more SystemC designs than IP-Xact models openly available, we considered extraction of IP-Xact structural representation from composite SystemC designs, our first contribution. To allow further model annotations beyond vendor extensions we also studied further transformations to UML MARTE profile. Next one should provide useful examples of such annotations.

Currently we have not extended the model extraction from SystemC to component behaviors themselves (as PinaVM does to some extent). As IP-XACT does not provide features for behavior modeling, this would have to target directly MARTE behavioral models (which then are amply defined, with hierarchical FSMs for control and data-flow activity diagrams for netlists. While it could be interesting to check whether extracted models there reflect the designer’s intuition, we strongly believe that proper design flow should produce efficient code representation in SystemC from MoCCs, not the other way around. Principles of abstraction/refinement (between RTL and various TLM sublevels), as well as timing level accuracy, would certainly find a more natural formulation at model level than code level.

**REFERENCES**


Abstract—The study of Networks on Chips (NoCs) is a research field that primarily addresses the global communication in Systems-on-Chip (SoCs). The selected topology and the routing algorithm play a prime role in the performance of NoC architectures. In order to handle the design complexity and meet the tight time-to-market constraints, it is important to automate most of these NoC design phases. The extension of the UML language called UML profile for MARTE (Modeling and Analysis of Real-Time and Embedded systems) specifies some concepts for model-based design and analysis of real time and embedded systems. This paper presents a MARTE based methodology for modeling concepts of NoC based architectures. It aims at improving the effectiveness of the MARTE standard by clarifying some notations and extending some definitions in the standard, in order to be able to model complex architectures like NoCs.

I. INTRODUCTION

System-on-chip (SoC) designs supply integrated solutions to challenging design problems of embedded systems and consumer electronic domains. Much of the progress in these fields allows the designer to conceive more and more complex electronic systems and reduce time to market. Future systems-on-chips will contain tens to hundreds of IP cores according to the International Technology Roadmap for Semiconductors [1]. The on-chip communication paradigm is introduced as Network-on-Chip (NoC) by Benini and De Micheli [2], Dally and Towles [3] and Sgroi et al [4]. Many propositions of NoC architectures for SoCs design have been presented in literature, such as, SPIN [5], HERMES [6] XPIPES [7], OCTAGON [8], Mesh [9], honeycomb [10].

Topology and routing algorithm are the two most important aspects that distinguish the various NoC architectures. The choice of topology is very important to provide a high level of performance. On the other hand routing on NoCs has the same principle as routing on any network. Consequently, the system becomes more and more complex, from transistors to layout. The system designer needs with a level of abstraction that focuses more on system functionality rather than low level design details and there is a demand for the development of high level modeling environments for computer-aided design. The modeling of highly repetitive structures such as network on chip topologies in graphical form poses a particular challenge. Model driven engineering is a software development methodology where the complete system is modeled at a high abstraction level using a modeling language as UML. The field of real-time embedded software systems is a domain for which extensions to UML are required to provide more precise notations of domain specific phenomena. The UML profile for Modeling and Analysis of Real-Time Embedded systems, MARTE [11] is the current standard for the SoC domain. We study in this paper how well it is suited to model complex NoC architectures.

We note that there is a few works interested in modeling concepts of NoC. Only the approaches defined in [12] [13] come close to our contribution by using the MDE concepts for modeling repetitive structure such as multi-processor system on chip (MPSoC) but not NoC concepts. In [12] the authors show the need for modeling the distribution of a parallel application onto parallel hardware architecture and describe the notation (MARTE) for modeling only regular architecture. This notation allows distributing computations to processing elements, data to shared or distributed memories, with the aim of clarifying its usage through MPSoC examples and the comparisons to other distribution notations such as in High Performance Fortran. We note that in literature there is a little work interested in modeling NoC with MARTE, defining methodology or providing the effectiveness for notation to model all concepts of NoC such as topology, routing algorithm, switching mode and communication protocol. The aim of this paper consists in proposing a novel methodology for modeling all family of NoC topologies and show that the advanced MARTE can support the modeling of routing algorithm with the use of state machines at the end to propose an extension in the HwCommunication package.

II. NETWORK ON CHIP CHARACTERISTICS

The NoC is a replacement for global interconnect and single bus architectures. The main immediate benefit of a NoC based
approach is clearly due to the possibility to reuse the communication network throughout different products. Furthermore, as the complexity of integrated systems keeps growing, a NoC provides enhanced performance such as throughout and scalability in comparison with the bus communication architectures. Thus it is useful to propose a methodology for modeling NoC topologies with the aims of being in line with the evolution of semiconductor technology and reduction of the time to market. Also, from a predictability perspective, the regularity of NoC layout provides well characterized electrical and physical properties. The topology and routing algorithm are the most important aspects which distinguish the diverse NoC architectures.

A. Topologies

The topology refers to the physical structure of the network graph, i.e., how network nodes are physically connected. It defines the connectivity or the routing possibility between nodes, thus having a fundamental impact on the network performance as well as the router structure (number of ports and port width). The trade-off between generality and customization is an important issue when determining a network topology. Each topology can be characterized by a few properties. The degree of a router is the number of links connecting that node to its neighbor vertices. A topology is considered as regular when all routers have the same degree, if not it is considered as irregular. In [14], the number of ports in routers can be synthesized according to the requirement of connectivity. However, the area and power consumption of an irregular network topology may not scale predictably with the topology size. Besides, many network topologies have been proposed where most of them are proposed for minimizing the number of nodes and node degrees. Also there are many research activities for designing NoC topologies. Murali in [15] have developed a tool for automatically selecting an application-specific topology for minimizing average communication delay, area, and power dissipation. Other topologies between regular and irregular are also proposed for NoCs. For example, an interesting NoC topology is the Spidergon of STMicroelectronics, GEXspidergon NoC [16] and Honeycomb [10]. Spidergon, is one of the NoCs researched at STMicroelectronics and is being proposed as an evolution of the STNoC [17]. It is a packet-switched NoC, inspired from the Octagon NoC [8], deterministic routing, wormhole switching and TDMA quality of service QoS. In [10], Hemani and all present a honeycomb NoC topology and its associated methodology as solution to the design productivity problem. Besides they propose a platform to handle the complexity of emerging design of chip architecture and still allow companies to meet the time to market and make profit, but implementations and results are not presented. Another study was made in order to define a generic NoC [18] named GeNoC that is intended to serve as a reference for the design and the validation of high level specifications of communicating virtual modules, but this study is limited to topology and routing concepts. The 2D-mesh is the most used topology due to its simplicity.

![Fig. 1. Examples of standard topologies](image1)

![Fig. 2. Examples of hierarchical topologies](image2)

It consists of horizontal and vertical lines with nodes placed at their intersections. This specific structure is often used because inter-node delays can be predicted at a high level.

Figures 1 and 2 show some of topologies that have been proposed in the literature.

B. Routing algorithm

Routing algorithms define the path followed by each message or packet routing on an NoC. A routing algorithm defines how the data are transmitted from sender to receiver. The choice of a routing algorithm depends on several metrics such as minimizing power, minimizing logic and routing tables to achieve a lower area, increasing performance by reducing delay and maximizing traffic utilization of the network. A lot of works aim at improving the hardware block for the routing algorithm in the NoC.

In this paper, routing algorithms are divided into two groups, deterministic and adaptive algorithms. Static routing algorithms ignore the network path diversity and are not sensitive to the network state. They are also simple and inexpensive to implement. Besides, it is often a simple way to provide the ordering of packets. Static routing also permits packets to be split among multiple paths between a source and destination, in a predetermined way. If only a single path is used, static routing usually guarantees in-order delivery of data packets. This eliminates the need for adding bits to packets at the NI (Network Interface), in order to correctly identify and reorder them at the destination.

Adaptive routing algorithms, also named dynamic algorithms, use information about network traffic and/or channel status to avoid congested or faulty regions of the network. In dynamic routing, routing decisions are made according to the current state of the network, considering a load on links. As such it is possible that the path between the source and destination changes over time, as traffic conditions and
requirements of the application change. This adaptive behavior requires additional hardware resources that control the state of the network and routing paths. Besides, dynamic routing is able to utilize alternate paths when some directions become congested.

Routing algorithms can be implemented in various ways. The most interesting ways consist of either looking at a routing table or executing a finite-state machine in software or hardware. In this paper we focus on the modeling of routing algorithms as a state machine to demonstrate the ability to model the routing on NoCs in MARTE.

### III. PROPOSED METHODOLOGY

Designing an efficient NoC architecture, while satisfying the application performance constraints, is a complex process. The design issues require several abstraction levels, ranging from high-level application modeling to physical layout level implementation. Some of the most important phases in designing the NoC include: NoC topology modeling for the application, choosing the routing algorithm, communication protocol and mode switching.

This methodology can be expressed in two phases. In the first phase, we begin by detailed concepts relating to a NoC, such as mathematical studies of topology and routing algorithm. Second, specify the package we will use and make the relation between hardware architecture concepts and notation that also exist in MARTE. We illustrate this below with the two main characteristics of NoCs identified above: topology and routing algorithm.

#### A. Topology Modeling

Several topologies, such as hypercube, star, ring and tree, can be modeled easily with the MARTE profile using the RSM (Repetitive Structure Modeling) package (see [11], annex E, pages 517–533). The more complex such as mathematical studies of topology and routing algorithm. This architecture is modeled on the same following algorithm to model GEXspidergon; in a customizable way on the environment GASPARD by using the RSM package, in order to take into account the modeling of link topologies. The concerned topology is composed of the repetition of a single element (Router or port). Each potential instance of this element is connected to other potential instances of the same element. In our case each instance is connected to neighbors located at north, south, east and west. The inter Repetition topology enables to specify the position of every neighbor of every potential instance of a model element with a multidimensional shape. With this architecture it is to the best of our knowledge the first time that a reshape connector is used with the same port as source and destination, allowing to represent regular but not uniform dependencies between repetitions of a part.

Figure 3 shows the modeling of honeycomb topology.

Mathematical models for Reshape and InterRepetition are described below:

- Inter Repetition=[1,0]
- Reshape include Tiler source and Tiler target:
- Pattern shape=\{0\}, Repetition space=\{0\}
- Source Tiler
- origin=\begin{pmatrix} 0 \\ 0 \end{pmatrix}, paving=\begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix}, fitting=\{0\}
Target Tiler
origin= \( \begin{pmatrix} 0 \\ 0 \end{pmatrix} \)  
paving= \( \begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix} \)  
fitting= \( (0) \)

2) Modeling of GEXspidergon Topology: The GEXspidergon (see Figure 2a) can be seen as a hierarchical and globally regular, locally regular topology. It is an academic topology for NoC. This study presents a generic NoC architecture based on a configurable router. This router integrates a sophisticated dynamic arbiter, the wormhole routing technique and can be configured in a manner that allows it to be used in many possible NoC topologies such as Mesh 2-D, Tree and Polygon architectures. This makes it possible to improve the quality of service (QoS) required by the proposed NoC. This study [17] shows that the Spidergon architecture is characterized by the lower latency and saturation. This architecture is constructed based on an elementary polygon network which is a combination of the star and the ring architectures. This elementary network is formed by \( 4R+1 \) \((R = 1, 2, \text{etc.})\) routers including a central router that is connected with the \( 4R \) peripheral routers via point to point links. The peripheral routers are connected to each other in the form of a ring. The elementary network is thus characterized by its valence \((m = 4R)\) that represents the number of the peripheral routers. These routers necessitate \( 2m \) links to be connected to the central router. Each peripheral router is connected to 4 input/output ports and the central router is connected to \( m+1 \) input/output ports.

This elementary network can be described in Parallaxis[19] as following:

**Configuration Poly** \([2...n],[2...m]\)

**Periphery connection:**

**Vertical connection:**
North: Poly\([i]\) → Poly\([i - 1]\)
South: Poly\([i]\) → Poly\([i + 1]\)

**Horizontal connection:**
East: Poly\([j]\) → Poly\([j + 1]\)
West: Poly\([j]\) → Poly\([j - 1]\)

**Central Router:**

Connection many to one
Poly\([i,j]\) → Poly\([0,0]\)

The GEXspidergon graph is constructed by iterating this algorithm in two dimensions as illustrated in Figure 2. Routers are categorized into four groups according to their degree:

- Corner Routers.
- Middle Routers.
- Horizontal Routers, package of two.
- Vertical Routers, package of two.

Figure 4 shows a possible model of this topology.

To express the link topology between vertical/horizontal routers and middle routers there was some problem since the considered tiles are not sets of regularly spaced point and in MARTE/RSM the points of the tile must be regularly spaced (as they are built from the reference point of the tile by the linear combination of the column vectors of the fitting matrix). We had to use two reshape links to express such a topology. These two reshapes are detailed below.

Reshape1: VR to MR

\[
\begin{pmatrix} \text{Source Tiler:} \\ \text{Target Tiler:} \end{pmatrix} = \begin{pmatrix} \text{origin} \\ \text{paving} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 0 \end{pmatrix} \\
\begin{pmatrix} 1 \end{pmatrix} \quad \begin{pmatrix} 0 \end{pmatrix} \quad \begin{pmatrix} 0 \end{pmatrix} \quad \begin{pmatrix} 1 \end{pmatrix} \\
\begin{pmatrix} 0 \end{pmatrix} \quad \begin{pmatrix} 0 \end{pmatrix} \quad \begin{pmatrix} 0 \end{pmatrix} \quad \begin{pmatrix} 1 \end{pmatrix} \\
\begin{pmatrix} \text{fitting} \end{pmatrix} = \begin{pmatrix} 0 \end{pmatrix} \quad \begin{pmatrix} 1 \end{pmatrix} \quad \begin{pmatrix} 0 \end{pmatrix} \quad \begin{pmatrix} 0 \end{pmatrix}
\]

With this case study we have shown that we can model the complex regular topologies with MARTE/RSM in a compact way, exploiting fully the regularity of the architecture of the NoC.

B. Modeling Routing Algorithms

In this paper routing algorithms, are divided into two groups, deterministic and adaptive algorithms. We are interested in modeling an example for each family.

1) Deterministic XY algorithm: The XY routing is considered as a kind of deterministic routing algorithms. It can be used in a \( 3 \times 3 \) Mesh topology NoC. In this architecture each router is identified by its coordinates \((x, y)\).

This algorithm begins by comparing the current router address \((C_x, C_y)\) to the destination router address \((D_x, D_y)\) of the data, stored in the header. Packet must attain the port
of the router when \((C_x, C_y)\) is equal to \((D_x, D_y)\) address. If not, \(D_x\) is compared to \(C_x\) address. Packet will be routed to the west port when \(C_x > D_x\), to east when \(C_x < D_x\) and if \(C_x \equiv D_x\) the packet is horizontally routed. When the last condition is true, the \(D_y\) address is compared to the \(C_y\). When \(C_y > D_y\) the packet is routed to the North but if \(C_y < D_y\) it will be routed to the South. Packet will be blocked in the Fifo when the chosen port is busy. This explication can be described with the following algorithm.

**Destination router:** \((D_x, D_y)\)  
**Current router:** \((C_x, C_y)\)  
If \((D_x > C_x)\) // Move East  
Return E  
Else if \((D_x < C_x)\) // Move West  
Return W  
Else if \((D_x = C_x)\) // same direction  
If \((D_y < C_y)\) // Move South  
Return S  
Else if \((D_y > C_y)\) // Move North  
Return N  
Else if \((D_y = C_y)\) //same router  
Return C  

Thus the routing algorithm can be modeled via MARTE profile as a hardware component where the behavior is described with a behavioral state machine in MARTE as shown in Figure 5.

2) **Adaptive routing:** In adaptive routing, paths are dynamically changed during the routing process according to network conditions, such as the presence of congestion. If a channel is busy, the other channel has priority over the busy channel. However, if both output channels are not used, a cost function (CF) decides which output channel will be used. This cost function is required when an adaptive routing is implemented. The adaptive routing algorithm increases the complexity of the hardware architecture of the router. That is why it will be significant to model this algorithm at a high level of abstraction. In this case a typical example of adaptive routing algorithm is modeled, the double Y-channel routing algorithm. This algorithm applies to a 2D mesh where router are connected by one bidirectional links in the X dimension and by two bidirectional links in the Y dimension. The network is divided into two sub-networks. The X1 sub-network uses paths in the ascending X dimension. The X2 sub-network uses paths in the descending X dimension.

We consider \(S\) and \(D\) the source and destination router of a packet. Their coordinates along the X axis are noted \(S_x\) and \(D_x\). If, at some router the destination of a packet is on the right the packet uses the X1 sub-network; if the destination is on the left the packet used the X2 sub-network. When \(D_x = S_x\) a cost function decides the channel to use. In each sub-network, several minimal paths are possible. A packet crosses the network through a single sub-network. This explication can be described with the following algorithm.

**Destination router:** \((D_x, D_y)\)  
**Source router:** \((S_x, S_y)\)  
If \((D_x > S_x)\) // Move in X1  
Return X1  
Else if \((D_x < S_x)\) //Move in X2  
Return X2  
Else if \((D_x = S_x)\) // CF  
Return CF  

This algorithm can be simply modeled in state machine via UML/MARTE profile as shown in Figure 6.

For the QoS there are a lot of work that integrate several services in the routing algorithm such as a dynamic arbiter, guaranteed throughput and TDMA. The designer can consider all these concepts in the modeling. On the other side, the switching technique can be an enumerated type in the MARTE profile.

The last thing that we need to model NoCs at a high level of abstraction is to decide which modeling artifact we will use to model a router. Indeed, this element is the basic building
block of NoCs and with which the topology can be described and that should be characterized by a behavioral state machine describing the routing algorithm. It should thus be a structured class with an adequate stereotype.

The HwCommunication package in the MARTE profile defines some concepts for modeling the communication infrastructure of embedded systems as « HwEndpoint » which may present the network interface of the NoC, « HwBridge » to make connection between resources and the « HwArbiter » to control the communication. But in this package the router (the basic building of NoCs as identified above) is not found. That is why we propose to add the « HwRouter » stereotype as a specialization of the « HwMedia » stereotype in the HwCommunication package. This addition would enable the modeling of NoCs at an abstraction level useful for specifying and analyzing the communication possibilities of the current (and probably future) networks used in recent systems-on-chip.

IV. CONCLUSION

We have proposed in this paper a methodology for modeling NoCs with the MARTE standard profile. We have identified two main characteristics of NoCs that are useful to design, analyze and understand the structure and behavior of NoCs: the topology of the network and the routing algorithm.

Several topologies have been proposed in the literature and we have shown that the Repetitive Structure Modeling package of MARTE is powerful enough to model as well the simple regular ones as the more complex ones. To achieve this, we have proposed the first use of the « Reshape » connector to connect some ports of the same part, enabling the description of regular but non uniform connections between repeated parts. The proposed modeling methodology enables to take full advantage of the regularity in the topology to factorize the model. Such factorization could then be used to generate factorized analysis or synthesis code and thus tackle large networks (think many-cores with several hundreds or thousands of routers) as easily as small ones.

The second characteristics that we have identified is the routing algorithm. We have proposed to model it with a standard UML behavioral state machine. This state machine should be attached to a component stereotyped « HwRouter », a new stereotype that we propose to identify the basic building block of NoCs. With this minor addition, the MARTE profile is complete enough to model a very large number of NoC proposals (actually, all those that we have found), especially those that exhibit some regularity enabling a scaling of the network. Some work is on-going to synthesize such networks in VHDL from such models.

REFERENCES


Modeling of Legacy Communication in Component-Based Distributed Embedded Systems

Saad Mubeen∗, Jukka Mäki-Turja∗† and Mikael Sjödin∗
∗Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden
†Arcticus Systems, Järfalla, Sweden
{saad.mubeen, jukka.maki-turja, mikael.sjodin}@mdh.se

Abstract

We propose the addition of special purpose component types to a commercially existing component model, the Rubus Component Model (RCM). The purpose of the new component types is to encapsulate and abstract the communications protocol and configuration in a component based and model based software engineering setting. With the addition of these new component types, RCM will be able to support state-of-the-practice development processes of distributed embedded systems where communication rules are defined early in the development process. We also show how an end-to-end timing model can be extracted from a distributed embedded system, modeled with RCM, to perform end-to-end timing analysis.

Keywords Model-Based Software Engineering; Component-Based Software Engineering; Distributed embedded systems.

I. Introduction

With the recent advancement in technology, embedded systems have become more and more complex. In order to deal with this complexity, lower development cost, reduce time-to-market and time-to-test, allow reusability and support modeling at higher level of abstraction, etc., the research community proposed the employment of Model-Based Engineering (MBE) and Component-Based Software Engineering (CBSE) for the development of embedded systems [1] [2].

Software development of distributed embedded systems is more complex as compared to single processor embedded systems. When MBE and CBD are used for the development of resource constrained and hard real-time distributed embedded systems, modeling of communication infrastructure arises as another challenge. The component model for the development of such systems should not only be resource efficient but it should also abstract the application software from the communication infrastructure. Moreover, it should also be able to model the legacy (previously developed) communications and legacy systems.

In this paper we propose the extension of a commercially existing component model, the Rubus Component Model (RCM) [3], by adding special purpose component types to it. RCM is a component model used for the development of resource constraint real-time embedded systems. It supports glue-code generation, end-to-end delay analysis, and resource requirements estimations. The purpose of the new component types, introduced in RCM, is to encapsulate and abstract the communications protocol and configuration in a component based and model based software engineering setting.

Our main goals in introducing these components are:

1) Allow model-based and component-based development of new nodes that are deployed in legacy systems that use predefined communications rules.
2) Support state-of-the-practice development processes where communications rules are defined early in the development process.
3) Enable adaptation of a node when communications rule change (e.g. due to re-deployment in a new system or due to upgrades in the communication system) without affecting the internal component design.

These goals are to be realized in RCM. The scope of this paper is PSMs (Platform Specific Models) for distributed embedded systems. With PSM we mean that the software component has been allocated to nodes and any adaptation to specific node characteristics (e.g. device drivers and memory layouts) has been added to the model.

Using our new components, nodes can be developed without explicit knowledge about the communication configuration.

Paper Layout

The rest of the paper is organized as follows. Section II presents the Rubus concept, the component model and its development environment. In section III, we present the related research and compare different modeling
approaches with ours. In section IV, we describe the new modeling objects to support modeling of legacy communication. Section V illustrates the extraction of timing model, from distributed embedded systems modeled with RCM, for end-to-end timing analysis. Section VI concludes the paper and section VII presents the future work.

II. Background – The Rubus Concept

The Rubus concept is based around the Rubus Component Model (RCM) [3] and its development environment Rubus-ICE (Integrated Component Environment) [4], which includes modeling tools, code generators, analysis tools and run-time infrastructure. The overall goal of Rubus is to be aggressively resource efficient and to provide means for developing predictable and analyzable control functions in resource constrained embedded systems.

A. The Rubus Component Model (RCM)

The purpose of the component model is to express the infrastructure for software functions i.e. the interaction between the software functions in terms of data and control flow. One important principle is to separate code and infrastructure, i.e., explicit synchronization or data access should all be visible at the modeling level. In RCM, the basic component is called Software Circuit (SWC). By separating code and infrastructure RCM facilitates analysis and reuse of components in different contexts (an SWC has no knowledge how it connects to other components).

The execution semantics of software components (functions) is simply:

1) Upon triggering, read data on data in-ports.
2) Execute the function.
3) Write data on data out-ports.
4) Activate the output trigger.

An example system in RCM is shown in figure 1. In this figure one can see how components interact with external events and actuators with regard to both data and triggering. Triggering events can consist of interrupts, internal periodic clocks or external events.

Furthermore, the component model has the possibility to encapsulate SWCs into software assemblies enabling the designer to construct the system at different levels of abstraction.

B. The Rubus Code Generator and Run-Time System

From the resulting architecture of connected SWCs, functions are mapped to run-time entities; tasks. Each external event trigger defines a task and SWCs connected through the chain of triggered SWCs (triggering chain) are allocated to the corresponding task. All clock triggered "chains" are allocated to an automatically generated static schedule that fulfills the precedence order and temporal requirements. Within trigger-chains, inter SWC communication is aggressively optimized to use the most efficient means of communication possible for each communication link (e.g. no use of semaphores in point-to-point communications within a chain, and sharing of memory-buffers between ports when there are no overlapping activation periods.)

Allocation of SWCs to tasks and construction of schedule can be submitted to different optimization criterion to minimize e.g. response times for different types of tasks, or memory usage. The run-time system executes all tasks on a shared stack, thus eliminating the need for static allocation of stack memory to each individual task.

C. The Rubus Analysis Framework

The model also allows expressing real-time requirements and properties on the architectural level. For example, it is possible to declare real-time requirements from a generated event and an arbitrary output trigger along the trigger chain. For this purpose, the designer has to express real-time properties on SWCs, such as worst-case execution times and stack usage. The constructed schedule will take these real-time constraints into consideration when producing a schedule. For event-triggered tasks, response-time calculations are performed and compared to requirements.

III. Related Research

There exist many component models for the development of distributed systems e.g. Distributed Component Object Model (DCOM) [5], Common Object Request Broker Architecture (CORBA) [6], Enterprise JavaBeans (EJB) [7] etc. These component models in their original form are not suitable for the development of resource constrained distributed embedded systems with hard real-time requirements because they require excessive amount...
of computing resources, have large memory foot print and have inadequate support for modeling of real-time communication. There are very few commercially existing component models for the development of distributed embedded and real-time systems especially in automotive domain. In the last decade, automotive research community and industry has focused more on the component based development of automotive embedded systems which led to the development of various solutions, approaches, methodologies, and models.

AUTOSAR (AUTomotive Open System ARchitecture) [8] is a standardized software architecture for the development of software in automotive domain. It can be viewed as a standardized distributed component model [9]. In AUTOSAR, the application software is defined in terms of Software Components (SWCs). The distribution of SWCs, their virtual integration and communication at design time is handled by Virtual Function Bus (VFB). Run-Time representation of VFB for each Electronic Control Unit (ECU) is defined by Run-Time Environment (RTE). The communication services are provided by the Basic Software (BSW) via RTE to the AUTOSAR SWCs.

When AUTOSAR was being developed, the main objective was to build a standardized infrastructure for automotive software development while handling of timing related information during the development was not considered. Furthermore, no focus was laid on specifying and handling of timing properties and real-time requirements during the process of system development. On the other hand, such requirements and capabilities were strictly taken into account right from the beginning during the development of RCM. AUTOSAR describes embedded software development at relatively higher level of abstraction as compared to RCM. A Software Circuit in RCM resembles more to a runnable entity which is the schedulable part of AUTOSAR SWC. As compared to AUTOSAR, RCM clearly distinguishes between the control flow and the data flow among SWCs in a node. AUTOSAR hides the modeling of execution environment whereas RCM explicitly highlights it.

In RCM, special interface objects (NOI and NII), which we will introduce in the next section, are used if SWCs require inter-ECU communication otherwise SWCs communicate via data and trigger ports. On the other hand, AUTOSAR does not differentiate between intra-node and inter-node communication at modeling level. Unlike RCM, there are no special components in AUTOSAR for inter-node communication. AUTOSAR SWCs use interfaces for all types of communications which can be of two types, i.e. Sender-Receiver and Client-Server. The Sender-Receiver communication mechanism in AUTOSAR is very similar to the pipe-and-filter communication mechanism used in RCM.

TIMMO (TIMing MOdel) [10] describes a predictable methodology and a language, TADL (Timing Augmented Description Language) [11], to express timing requirements and timing constraints during all design phases in the development of automotive embedded systems. TIMMO development methodology makes use of structural modeling provided by EAST-ADL [12] which is a standard architecture description language to model a system at various levels of abstraction. The model structure of TIMMO abstracts the modeling of communication at implementation level (defined by EAST-ADL) where the methodology proposes to use AUTOSAR. Both TIMMO methodology and TADL have been evaluated on prototype validators. To the best of our knowledge there is no concrete industrial implementation of TIMMO. In TIMMO-2-USE project [13], the results of TIMMO will be further validated and brought to the industry.

Object Management Group (OMG) defined middleware technology such as Real-Time CORBA, minimum CORBA and CORBA lightweight services for the development of real-time and distributed embedded systems [14]. Real-Time CORBA has been used to develop distributed embedded systems [15] [16]. Because of higher resource requirements, these models may not be suitable for the development of resource constrained distributed embedded systems with hard real-time requirements.

COMDES-II (COMponent-based design of software for Distributed Embedded Systems) provides a component-based framework for the development of distributed embedded control systems [17]. It uses labeled (named) messages for the network communication. The scheduling policy used by OS in COMDES-II is fixed priority timed multitasking scheduling. On the other hand, Rubus Operating system implements hybrid, static and dynamic, scheduling without using timed multitasking [18].

IV. Support for Legacy Communication

In an ideal scenario, it should be possible to automatically generate the communication for each application from the design model. However, this is often not the practice in the industry because there exist legacy communications and legacy systems. These systems have their own predefined rules of communication. Our goal is to introduce the support for modeling of legacy communications in RCM.

To support abstraction of the implementation of communications in a node, we propose the introduction of two special purpose modeling elements: the Network Input Interface (NII) and the Network Output Interface (NOI). In order to represent the model of communication in a physical bus, we propose another modeling object called Network Specification (NS). In this section we describe these three new modeling objects in detail.

A. Network Specification (NS)

It is the model representation of a physical bus. There are two parts of NS. One is protocol independent and the other is protocol dependent. The protocol independent part defines a message and its properties such as ID, sender node ID, list of receiver nodes IDs,
B. Network Input Interface (NII) Component

It is the model representation of incoming signals from the network. Thus, NII component describes all signals that can be received by a node from a network. There is one NII component per network that the node is connected to. Associated to each signal is a data-port and a trigger-port. When a frame arrives at the node, the physical bus driver and protocol specific implementation of the NII extract the signals (zero or more signals per frame) and encode their data in the RCM data-type. When the signal(s) is delivered, the data is placed on the corresponding data-port and the trigger-port is triggered. If the trigger is not used then it can be connected to the ground. Figure 2 graphically illustrates the NII component.

C. Network Output Interface (NOI) Component

The NOI is a component that describes all signals which can be sent from a node on a network. Hence, it is the model representation of the outgoing signals on the network. There is one NOI per network that the node is connected to. The major difference from the NII is that the NOI does not have any trigger ports. Conceptually, the NOI has an implicit trigger port for each data port—however, to lessen the burden for the modeler these ports are omitted from the model. The NOI uses protocol specific rules on how to map signals to frames and encode data in the frames. The NOI also uses protocol specific rules to decide when to send each frame. Thus, the SWCs that use the NII are kept unaware about details such as signal-to-frame mapping, data-type encoding, and transmission patterns (common transmission patterns are: periodic, on-change, on-change with minimum distance between messages, etc.). Figure 3 graphically illustrates the NOI component.

Both NII and NOI components can be automatically generated from NS by a Network Configuration Tool. This tool also carries out mapping between NS and network interface components and vice versa. The network interface components are translated into a set of SWCs to execute the protocol at run-time. Figure 4 graphically illustrates the model of network communication with new modeling components in RCM.

Fig. 2. Graphical illustration of NII

Fig. 3. Graphical illustration of NOI

Fig. 4. Model of NOI and NII in a node

Analyzability was one important aspect that was kept in mind while introducing the new component types in
RCM. The objective was to enable RCM to not only model the legacy communication but also to analyze the end-to-end timing behavior of the modeled system. In the next section we will discuss how the required timing information is extracted from a distributed embedded system, modeled with RCM, to perform end-to-end timing analysis.

V. Extraction of Timing Model for End-to-End Timing Analysis

In real-time systems, the time at which the result is available is as important as the correct result. With newly introduced modeling elements in RCM, we can model a complete distributed real-time embedded system. In order to ensure that all timing requirements are met, the modeled system should render itself to end-to-end timing analysis. To perform the timing analysis, end-to-end timing model of the modeled system should be available. The computation model for timing analysis considered in this paper consists of a task model with offsets [19] [20] [21] and a communication model [22]. The task model is used for response-time analysis of tasks in a node whereas, the communication model is used for response time analysis of messages in the network. We illustrate how we can extract end-to-end timing models for distributed transactions modeled with the new modeling elements in RCM. From the extracted model, we will analyze the end-to-end timing for delays and network utilization.

In order to understand which timing information needs to be extracted for end-to-end timing analysis, we consider an example. Figure 5 shows a block diagram of an example distributed embedded system modeled with RCM using the new modeling objects. There are two nodes in the system with three SWCs per node. SWCs communicate with each other using both inter-node and intra-node communication. The inter-node communication takes place via Controller Area Network (CAN) to which the two nodes are connected. An event chain (distributed transaction) that consists of four Software Circuits i.e. SWC1, SWC2, SWC4 and SWC5 is identified with bold lines in figure 5. In this transaction, an event triggers SWC1 which in turn triggers SWC2. SWC2 then sends a signal to NOI which in turn maps it to a CAN frame. This frame is transmitted over CAN bus and is received by the NII of the receiver node. The NII decodes the signal and places the data on the corresponding data port and it also triggers the corresponding trigger port. The elapsed time between the arrival of the triggering event at the input of SWC1 and the instant when SWC5 gives response is referred to as end-to-end delay of the distributed transaction and it is shown in Figure 5.

The end-to-end timing model should contain timing related information of all transactions in the system. At node level, the timing information includes the total number of tasks in the system, Worst-Case Execution Time (WCET) of each task, trigger Period (periodic activation) or inter-arrival time (event activation) between successive events triggering a chain, Jitter etc. At network level the timing information is specified in NS. It includes bus speed, number of frames, frame transmission time, frame period (periodic frame) or inter-arrival time (event frame) or both (mixed-type frame), frame jitter etc.

In Rubus-ICE, when the designed model is completed it is compiled to Intermediate Compiled Component Model (ICCM) file [23]. All the timing information required by the end-to-end timing model is extracted from ICCM file. From this timing model, Rubus analysis framework performs response-time analysis of individual tasks [21], response-time analysis of messages on the network [22] [24] and end-to-end timing analysis [25]. The analysis framework provides the results i.e. response-time of individual tasks, response time of frames, end-
to-end delay, network utilization etc. back to Rubus-ICE. This whole process is depicted in Figure 6.

VI. Conclusion

We introduced new modeling elements in a commercially existing component model for the development of distributed embedded systems. The purpose of the new component types, Network Input Interface and Network Output Interface, is to abstract the implementation and configuration of communications in distributed embedded systems. The components make the communications capabilities of a node very explicit, but efficiently hide the implementation or protocol details. We also demonstrated the extraction of end-to-end timing model from a distributed embedded system modeled with RCM.

VII. Future Work

In future work, the implementation of NII and NOI will be automatically generated from protocol configuration files, like CANopen, DCFs (Device Configuration Files) or for subsets of J1939.

Acknowledgement

This work is supported by Swedish Knowledge Foundation (KKS) within the project EEMDEF. The authors would like to thank the industrial partners Arcticus Systems and Hägglunds BAE Systems for cooperation.

References


On the Need for Extending MARTE with Security Concepts

Mehrdad Saadatmand∗, Antonio Cicchetti∗, Mikael Sjödin∗
∗Mälardalen Real-Time Research Centre (MRTC)
Mälardalen University, Västerås, Sweden
{mehrdad.saadatmand, antonio.cicchetti, mikael.sjodin}@mdh.se

Abstract—Security has often been considered as an added feature to the design of systems and this has been the root of many security issues. The need for introducing and considering security along with other aspects of the system from early design phases has now gained much more attention and been the subject of many studies. In model-driven engineering, this has also led to the introduction of several modeling languages for security. However, in embedded systems domain due to the limitations of resources, security requirements and features cannot be considered in separation as they impact other aspects such as schedulability, performance and power consumption. MARTE as a modeling language for real-time embedded systems contains the concepts for modeling these aspects, and hardware and software entities. In this paper, we discuss potentials and suitability of MARTE for extension with security concepts and how it can serve as a common framework for modeling security in embedded systems and performing trade-off analysis with other aspects.

I. INTRODUCTION

Considering security as addition of features to a system, not only can lead to inefficient and not optimal use and integration of security mechanisms, but can also impair the design and quality of the system. This issue is more critical in embedded systems that require careful balance among different properties due to resource constraints. Therefore, security should be considered from early phases of design and as a new dimension and metric [1]–[3]. In model-driven engineering which can help with raising the abstraction level and cope with the design complexity of embedded systems, several modeling languages for security have been defined to bring security concepts into the design models of the system. However, due to the characteristics of embedded systems, as will be discussed, security requirements cannot be considered in separation from other requirements. In order to achieve this, modeling language(s) used to design the system should be able to cover these variety of requirements.

MARTE [4] has set a firm ground in modeling non-functional requirements in real-time embedded systems. Considering that MARTE provides concepts for modeling generic non-functional requirements and also has dedicated (sub)profiles for modeling timing, software and hardware resources, we believe security deserves special attention in MARTE for the following reasons:

• The awakeness and shift towards considering and bringing security aspects in higher levels of abstraction and system design.
• Security in embedded systems is of great importance. While security problems in desktop applications could lead to problems such as personal information leaks and financial issues (e.g. credit card, bank systems), in embedded systems they could cause more serious issues such as injuries, death, enemy threats and so on; e.g. in medical systems, military equipments, power plant controllers, public water monitoring systems.
• Security aspects in embedded systems should be modeled along with other non-functional requirements as they incur big impacts and require balance and trade-off analysis with other aspects such as performance, resource usage, availability and schedulability.
• MARTE already has concepts that can be used as basis for definition of security features.

In summary, the appearance of different UML profiles for modeling security in recent years, ever-increasing importance of security in embedded systems, and potentials of MARTE for modeling security (e.g. using concepts in Non-Functional Properties (NFP), Generic Resource Modeling (GRM), Generic Quantitative Analysis Modeling (GQAM), Hardware Resource Modeling (HRM)) all put MARTE in a good position to move towards supporting security modeling. By extending MARTE with security concepts, it becomes possible to have a more precise system model and thus even in case of schedulability and performance analysis which MARTE explicitly supports, a more accurate analysis will be possible (i.e. taking into account impact of security measures).

In this paper, we investigate the benefits of extending MARTE with security aspects and why MARTE is a good choice to build security concepts upon. In section 2, importance and complications of security requirements in embedded systems are discussed. A brief look on security modeling is also offered and potentials of MARTE to incorporate security aspects are highlighted. Section 3 pictures an example of an embedded system where MARTE to incorporate security aspects are highlighted. Section 3 pictures an example of an embedded system whose security requirements directly impact other requirements and properties of the system. In section 4 and 5 we focus on encryption and authentication requirements for security in this system and describe our suggested approach to model these features based on MARTE and how it facilitates trade-off and sensitivity analysis. Section 6 focuses on the issue of incorporating security extensions in MARTE and its challenges. Finally in section 7, we give a summary of the
issues covered in the paper and explain future works.

II. MOTIVATION AND POTENTIALS

A. Security in Embedded Systems

There are additional challenges in designing secure systems when it comes to embedded systems domain. Embedded systems are supposed to operate as part of other systems (e.g., in vehicles, medical devices, etc.) and they can be used in hostile environments which make them prone to a variety of physical security threats. Also, the big increase in the development of connected embedded devices and their operation in distributed networks require new security considerations during their design. Besides security requirements originating from the usage and operation environment of embedded systems, [1] also discusses other security design challenges unique to embedded systems: security processing gap, assurance gap, battery gap and [security mechanisms] flexibility issue. By looking at these unique challenges from a higher level, we can realize that these requirements mostly originate from the nature of embedded systems which is being constrained in resources such as limitations on power consumption, processing capacity, maintainability, layout and physical dimensions. This leads us to the issue of trade-off between different non-functional properties of embedded systems. For instance, the flexibility issue mentioned in [1] is actually the trade-off between security and maintainability and upgrade-ability of the system. Similarly, battery gap is the trade-off between security mechanisms and available power resources.

B. Security Modeling

There are several efforts on defining UML profiles for security. For example, SecureUML [5] focuses on modeling role-based access control. AuthUML [6] provides a framework for analysis of access control requirements. [7] introduces a set of stereotypes for specification of vulnerabilities that serve as guidelines for developers to avoid them during implementation. UMLsec [8] offers a broader range of security concepts and comes with an analysis tool. [9] tries to offer a solution for modeling security along with timing characteristics of the system using UMLsec and MARTE.

One issue with such profiles and their usage is that most of them are limited in the sense that they usually focus on a certain aspect of security [10]. Therefore in modeling embedded systems especially when they are distributed, we need to apply several different security profiles to cover aspects like authentication, key-exchange, encryption, decryption and access control. This can sometimes be tricky considering that these profiles can have overlapping and conflicting semantics and notations. Also modeling of security requirements is often considered in separation from other requirements such as timing [9], [10]. Besides the ongoing efforts on modeling security, impact analysis and relation of security to other properties of the system is also a challenge.

Another issue with definition and use of separated security profiles is that it is not always straightforward to use a combination of profiles to cover different design aspects of a system; for example, using MARTE and SysML plus SecureUML. Specification and semantic conflicts can occur when combining different profiles [11].

C. Hardware Security

The environments that embedded devices are used in, make them more prone to security issues and types of attacks that are less relevant for other systems. While software security measures for a central database, for example in a company or university, are obviously a necessity, the situation for embedded devices is more complicated. For embedded devices such as flash drives and also mobile phones that can be carried around and contain huge amount of sensitive information from private user data (e.g., billing information for mobile Apps) to manufacturer firmwares, operating systems, and confidential algorithms and codes, security measures more than just software level are required. Therefore, in the design of embedded systems, security measures for physical and side channel attacks (including timing analysis, power monitoring, fault induction and electromagnetic analysis) and requirements on tamper-resistant hardware should also be considered [1]. [2].

Considering these issues and also resource limitations in embedded systems, dedicated hardware for security control and also hardware parts with built-in security support are attracting more attention than before. There are manufacturers that design cryptographic hardware accelerators and custom CPUs for lower power consumption and high-performance devices [1], [2], [12]. Advantages and disadvantages of having hardware intrinsic security and awareness of its use in industry are mentioned and surveyed in [13]. Therefore, in designing a secure embedded system, hardware aspects should also be taken into account. However, few security modeling solutions offer support to cover hardware security requirements in the system. Part of this problem could be due to the need to include concepts for describing hardware in such modeling languages. Specifying hardware units with built-in security support can affect allocation and deployment scenarios and thus is important to be modeled. On the other hand, MARTE already includes basic concepts for modeling hardware platforms and resources in embedded systems as well as allocation and deployment. This again shows potentials of MARTE as a common and unifying framework for adoption and definition of security concepts in embedded systems.

D. Potentials of MARTE

The rich concepts in MARTE for specification of non-functional properties, modeling of time, allocation and platform, and also support for schedulability and performance analysis give MARTE good potentials to answer security issues described above especially regarding the unification problems mentioned in [9] and [10] and offering a single framework. From this point of view, use of MARTE is a promising approach for cross-cutting nature of security aspects and providing a concise model of the system without redundant information modeling. For example, by extending
MARTE with security concepts, one use case could be to define secure ports and connections for components in MARTE component model, and this way highlight secure data flow and encryption requirements in the system and disallow designs that breach security (secure flow of data to an untrusted component). Without using a unified approach, several elements may have to re-appear in different models in order to cover various aspects of the systems; for example one to model security, one to model allocation and so on. Another example in which an extended version of MARTE (with security) proves beneficial is in modeling embedded systems which use hardware that have security support. Also as mentioned above, there are families of boards and microcontrollers which have hardware implementations of cryptographic algorithms. MARTE Hardware Resource Modeling (HRM) profile can be extended to enable modeling of such systems. An interesting work which tries to add security modeling to MARTE is [10]. In this study, necessary concepts for modeling and analysis of resilience (as a security feature) are defined as a profile called Security Analysis and Modeling (SecAM). This suggested profile is based on MARTE profile for Dependability Analysis and Modeling (DAM) [14].

III. Motivation Example

To show the benefits and applicability of an extended version of MARTE (MARTE+Security), we use the automotive example that is mentioned in [15]. This example is a desired use case for electronic payment systems that can be embedded in vehicles. In this case, a vehicle enters a parking garage. The vehicle starts interacting with the garage to receive information on parking cost, notifies the driver about that, and upon exit pays the fee automatically through an authorized third party (e.g. vehicle OEM). These interactions are shown in Figure 1. To design such a system, different aspects such as schedulability, performance, allocation and security should be considered.

To model the security aspects of this system, SecureUML is not enough since it only focuses on access control and lacks modeling support for issues like encryption and decryption mechanisms. Another security profile such as UMLsec [8] can be used which covers a wider variety of security concepts. However, as mentioned before, modeling security aspects in embedded systems separately can be problematic. For example, by introducing such electronic payment system in vehicles, there is now a tighter relation between security and other requirements of the system. One problematic scenario is the relation between schedulability, performance and security. If the security protocols that are used require heavy computations and are not well designed, other tasks in the system may miss their deadlines. It can be from a simple window closing task to automatic braking or central lock system which in the end affect safety requirements of vehicles. An extreme case of this situation could be when the vehicle is busy performing the payment, and the driver needs to close the windows or lock the doors due to a burglary threat inside the garage.

Also for OEMs to distinguish between vehicles, perform transactions and issue billing information accordingly, a mechanism is required to uniquely identify each vehicle. This mechanism should be immune to impersonation attacks, otherwise malicious parties (e.g. driver of the vehicle) may try to have other people/vehicles’ accounts charged.

IV. Modeling Encryption and Enriching Analysis

As discussed, satisfying security requirements comes with a cost in terms of performance, energy and memory consumption and impact on other properties of the system. In order to consider this cost in the design of systems, it is needed to know specifications of applied security mechanisms such as encryption. In this section, we focus on the encryption requirement of AuthorizePayment() operation and show how modeling this requirement using MARTE enables performing sensitivity and trade-off analysis.

To model this security requirement we have defined a stereotype called Encryption and ’specialized’ from it different types of encryption particularly block ciphers that are used in this example. Figure 2 shows definition of this stereotype based on MARTE NFP concepts.

As AuthorizePayment() involves message transfer between Vehicle and OEM, it is labeled as CommunicationStep (GaCommStep or SaCommStep depending on the Analysis Context) [4]. First, applying our stereotype, AuthorizePayment() is specified as:

\[
\langle\text{BlockCipher}\rangle\quad \text{AuthorizePayment()} \quad \{\text{algorithm=AES}, \quad \text{blockSize=(128,bit)}, \quad \text{keySize=(128,bit)}, \quad \text{rounds=12},
\]
Fig. 2. Definition of BlockCipher stereotype

operationMode=ECB

In a schedulability scenario, SaCommStep from Schedulability Analysis Modeling (SAM) profile can then be applied to support schedulability analysis:

\[ \langle\langle\text{SaCommStep}\rangle\rangle \langle\langle\text{BlockCipher}\rangle\rangle \text{AuthorizePayment()} \{\text{algorithm=AES} , \ \text{blockSize=(128,bit)}, \ \text{keySize=(128,bit)}, \ \text{rounds=12, operationMode=ECB}, \ \text{msgSize=(150,B)}\} \]

Having the above details specified about the encryption mechanism in the model enables us now to incorporate the results of studies such as [16] and [17]. In [17], execution times for different encryption algorithms based on input size have been measured. [16] investigates energy consumptions of block cipher encryption algorithms with different settings (e.g. rounds, mode of operation, etc.). It also includes some comparisons of software versus hardware encryption. Figure 3 and 4 show examples of these results.

Fig. 3. Execution times of block cipher algorithms in ECB mode of operation [17]

Fig. 4. Energy consumption of block ciphers for different data lengths [16]

Since the security-performance analysis only knows about security concepts it only calculates the execution time for the encryption part of AuthorizePayment operation and the total execution time of this operation will be more than this value (hence using ‘min’ as StatisticalQualifier). In cases where energy consumptions of encryption algorithms are also important such as in Wireless Sensor Networks and in order to perform an energy consumption analysis on the model, energy values can be calculated and included in the model analysis context:

\[ \langle\langle\text{GaCommStep}\rangle\rangle \langle\langle\text{BlockCipher}\rangle\rangle \text{AuthorizePayment()} \{\text{algorithm=AES} , \ \text{blockSize=(128,bit)}, \ \text{keySize=(128,bit)}, \ \text{rounds=12, operationMode=ECB}, \ \text{execTime=(330,ms,min,calc)}, \ \text{energy(0.23,mj)}\} \]

Knowing timing and energy values based on the chosen encryption method and input message size, sensitivity analysis on the model is possible now to determine the best trade-off between security, timing and energy consumption. For example, if it is realized that the execution time (and/or energy consumption) is too much, the message to be encrypted can be reduced in size, parameters of encryption algorithm may be changed or another encryption method can be selected instead. It is important to note that these evaluations are now feasible before implementation and reaching the code level.

V. VEHICLE AUTHENTICATION

In the automatic electronic payment system, in order for the OEM to distinguish between different vehicles, several mechanisms can be used. One way could be to issue a smart card for each owner and have an embedded card reader in the vehicle to read the card information and send it for identification. Another possible solution is to use a unique identifier for each vehicle. For instance, registration plate number of the vehicle can be stored in a memory chip and retrieved and sent over to the OEM to identify the vehicle.

To store unique identification information securely, a secure memory module can be used (refer to [18] for examples of available secure memory modules). To be able to include this scenario in the system, a stereotype for secure memory modules is defined. Building upon hardware resource modeling
VI. INCORPORATING EXTENSIONS IN MARTE

One step in adding security to MARTE is to apply an appropriate taxonomy for security concepts as there are different suggested classifications in this area. Definition of security concepts in a way that the model can be used (e.g. through transformation) as input model to security analysis tools is also an important issue that should be taken into account. For example, Encryption concepts we defined in this work can be transformed to Secrecy concept of UMLsec if security aspects of the model are to be analyzed by UMLsec analysis tool.

To offer security concepts as extension of MARTE, we are working on defining an appropriate structure for the profile considering different classification and taxonomies for security that will also be in line with MARTE’s way for grouping of concepts into packages and subprofiles. Figure 6 shows the current structure of our suggested profile.

![Diagram of suggested structure for Security profile]

Fig. 6. Suggested structure for Security profile

Logical Security in this structure will consist of the following packages: Secrecy Confidentiality, Integrity, Authentication, Authorization, Access Control, Non-Repudiation and Data Freshness. Hardware Security will include concepts for modeling hardware intrinsic security and physical measures to protect and secure embedded systems, e.g. against side channel attacks and making systems tamper-resistant. To group concepts such as vulnerability, threat, attack, attacker and intrusion, Security Incident package is defined which is inspired by studies such as [19].

VII. CONCLUSION AND FUTURE WORK

There are efforts on defining modeling languages for security, however, when it comes to systems such as in embedded domain where system properties are tightly interconnected, security requirements cannot be considered in isolation from other requirements such as timing and energy consumption. In this paper, we discussed this issue and how MARTE can serve as a common framework to build security concepts upon. Strong features of MARTE toward this goal were highlighted and it was shown how an appropriate specification of security can help to perform sensitivity and trade-off analysis among requirements on the model. The latter is especially important in embedded systems domain where resources are limited.

Including security concepts and dedicating packages and profiles for them in MARTE, can also help with raising the awareness of system designers towards considering and including security decisions in design models. This explicit support for security in the modeling language helps with the problem of considering security as an afterthought and added feature to the system. Especially that you cannot almost build an embedded systems these days with no security.

As a future work it is interesting to try to generate code with security features from MARTE+security models. For example, Encryption and BlockCipher stereotypes and their respective properties can help with determining the right encryption algorithm to implement and generating the code for it. Successful generation of code from these specifications and its generation percentage and coverage is left to be investigated and tested as a continuation of this work. Also using MARTE+security models as inputs to security analysis tools to evaluate security requirements and security level of the system is another future direction of this paper.

REFERENCES


