

# Executable SystemC specification of the MARTE generic concurrent and communication resources under different Models of Computation

Pablo Peñil<sup>1</sup>, Eugenio Villar<sup>1</sup>, Héctor Posadas<sup>1</sup>, and Julio Medina<sup>2</sup>

<sup>1</sup>Microelectronics Engineering Group, Universidad de Cantabria, 39005-Santander, SPAIN  
{pablop, villar, posadash}@teisa.unican.es

<sup>2</sup>Departamento de Electrónica y Computadores, Universidad de Cantabria, 39005-Santander, SPAIN  
julio.medina@unican.es

## Abstract

Modeling and analysis of real-time, embedded systems is becoming an important area of research nowadays. In this context, the UML MARTE profile has been proposed to support the specification, design, and verification stages in the design process. It provides a wide set of facilities to introduce all the information required in the first steps of the design process. To carry out the actions involved in these design steps, MARTE-based tools are required. This paper presents a methodology to automatically generate SystemC executable specifications from generic MARTE models. The methodology proposed is based on the SystemC language. SystemC is a C++ extension, widely used in the electronic design community to generate executable specifications of embedded systems. To generate these specifications, the information regarding the system structure and hierarchy is extracted from the MARTE models. A subset of concurrency and communication features of the MARTE profile is used for this purpose. Thus, semi-automatic generation of the executable specifications is possible. Then, the code required to describe the system operations can be easily integrated into the executable code. This design methodology establishes a refinement flow to perform the design steps before deciding system partitioning.

## 1. Introduction

The evolution of electronic technology has enabled the integration of more components in a single chip. The increasing quantity and complexity of the system components is a major factor making the design of embedded systems more difficult. Current embedded system architectures include a wide variety of components of different nature: general purpose processors, application-specific HW, reconfigurable HW, DSPs, ASIPs, etc. As a consequence, heterogeneity is necessary to deal with such a variety of components. This heterogeneity has to be handled during the entire design process, especially during the first development steps.

Common electronic design flows start by creating abstract models of the system to be implemented. The abstract model is a platform-independent model (PIM) that depicts the general functionality of the system. After that, design flows continue with the generation of an executable specification to validate the functionality of this initial specification.

When creating the abstract system models and the corresponding executable specifications, different descriptive mechanisms are required. The description of each component depends on its internal semantics and the interconnection with the rest of system.

To handle these semantics, we must start considering that electronic embedded systems are massively concurrent. Electronic systems are not only composed of an increasing number of concurrent components but also each component, independently of its HW or SW nature, is itself composed of a large number of concurrent elements: gates, in the case of HW and threads in the case of SW. Beyond the number of gates or threads, concurrency is the main reason for the increasing complexity of components.

To model these semantics rigorously, the models of computation (MoCs) covered in all through the system have to be considered. The characteristics of the components and the interaction among them should be understood under a certain Model of Computation (MoC). Furthermore, since all system components are highly interconnected, it is also necessary to ensure the integration of the different MoCs. Using the appropriate MoC, it is possible to describe the system with deterministic mechanisms and also provide additional advantages [1].

The goal of this paper is to apply MoCs on UML/MARTE descriptions to allow automatic generation of executable specifications in SystemC. This guarantees correct, fast validation of the UML/MARTE models.

The proposed methodology provides the designer with a set of guidelines to describe the system under several models of computation (MoC) using the MARTE profile [3]. The UML profile for MARTE has been developed to model and analyze real-time systems, providing the concepts needed to describe real-time features that specify the semantics of these kinds of systems at high abstraction levels. Additionally, the semantics of the real-time concepts are clearly and precisely defined by MARTE, which enables an executable model to be obtained from the system model.

The paper is organized as follows. In section 2, related work is analyzed. Section 3 describes the methodology and design flow proposed. Section 4 presents the formal mechanisms proposed to describe the system considering several MoCs. In section 5 the MARTE elements selected to describe the characteristics of each MoC are presented. Section 6 explains how the elements presented in section 5 should be used to describe heterogeneous system models. In addition, it relates the mapping among the UML/MARTE elements and the SystemC elements to generate heterogeneous executable specifications based on the HetSC methodology. Finally, sections 7 and 8 draw some conclusions and establish future working lines.

## 2. Related work

UML and SystemC have been used as system design languages since the first appearance of UML [8]. Several proposals for bridging the gap between UML specifications and SystemC

executable models have been published. The first area for combining UML and SystemC was using UML stereotypes for SystemC constructors. This combination was focused on a system-on-chip (SoC) design methodology such as [25]. With the work of Riccobene et al. a SoC design flow was proposed based on a SystemC profile [9, 10, 11] used to produce an executable model from a UML specification. In the methodology presented in this paper, SystemC is used to create an executable specification of a model expressed in UML/MARTE to validate it; the system model is only created using UML/MARTE concepts. Alternatively, the Riccobene et al. work uses SystemC as a model language of the system. They created a system model with the SystemC semantics. Furthermore, the use of a SystemC profile enables a HW/SW co-design approach [12] and in [13] there is an overview of UML to cover the SoC and hardware-related embedded system design. In this methodology, the partition between software and hardware is not considered. This decision is postponed to later design stages. Another way of combining UML and SystemC is using mapping rules [20] to establish an automatic transformation between UML and SystemC. This work provided the first ideas about the mapping among the UML elements and the SystemC elements. The relation between a port with a provided interface and the SystemC port `sc_export` is taken from this work. Apart from that, additional ideas were taken into account in the initial development stages of this methodology, but some of them were ruled out due to the rules that HetSC methodology establishes. In [21] a mapping between UML application models and the SystemC platform models is proposed in order to define transformations rules to enable semi-automatic code generation. This mapping is used in a case-study [22]. Apart from that, SysML [7] has its own generation flow to SystemC. In [23] [24], a mapping between SysML and SystemC constructs is described to automatically generate an executable model.

From the appearance of the MARTE profile as a design language, several works have been published to show the capabilities and the advantages that this new profile provides at different design flow stages. Taking MARTE-based models, the methodology proposed in [14] describes embedded real-time applications. The model is transformed into an executable platform through the code generation Accord/UML [15][16]. The transformation of MARTE specifications to SystemC executable models is explained in [17]. The application of a MARTE methodology for hardware modeling is proposed in [33]. In addition, in [34] MARTE is used for hardware modeling as well, but combining MARTE with a new profile, namely ESL profile based on IP-XACT concepts. Gaspard2 is a design environment dedicated to MPSoC for moving from the high-level MARTE description to a SystemC executable platform for data-intensive applications. Gaspard2 can be applied in a design flow to obtain post-partitioning executable models. On the contrary, the methodology presented in this paper faces the design of the system from an abstract view, that is, only concurrent elements are identified. It allows the design of the system to be explored under several approaches at pre-partitioning stages.

An analysis of how executable models can be generated from SysML and MARTE is shown in [18]. SysML contributes with the structure concepts and MARTE adds the time modeling [19]. Nowadays a subset of UML [8] is being developed to provide the

concepts needed in order to specify executable models that are executed in a virtual machine.

### 3. Design Flow

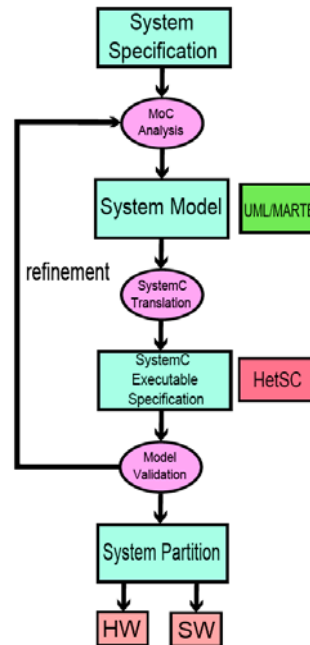


Figure 1. Design flow.

This paper presents the first stages of a design methodology based on the MARTE profile. The design flow is shown in figure 1. The MARTE profile is used to model a heterogeneous system where each part of the system has a specific relation with the other system parts. The relation among the different parts in the system is understood under the communication semantics of a certain MoC. Once the model system has been created, the next step is a transformation process; the system model is transformed into an executable specification. The executable specification is generated in SystemC [5], under the guidelines provided by the HetSC methodology [4] [28]. HetSC is a methodology for the specification of concurrent heterogeneous embedded systems in SystemC. The HetSC methodology does not distinguish between application level and platform level; it only identifies concurrent elements and communication instances. This characteristic makes HetSC different compared to other SystemC methodologies such as TLM, where its application domain is to specify platform models. The executable specification simulates the system model and, thus, the functionality is checked. Additionally, this design flow establishes a refinement process. The initial system model may be created from an ideal view. Once this initial model is simulated, it is possible to introduce several restrictions and new features that provide a more real description of the system. This refinement flow is carried out by expressing different MoCs since the election of a specific MoC depends on the functionality to be expressed and the level of abstraction.

## 4. Formal Background

The starting point of this methodology is ForSyDe. ForSyDe (Formal System Design) [1][2] provides a formal notation to describe a system at the highest abstraction level. The ForSyDe metamodel identifies concurrent processes communicated by signals. Taking the ForSyDe abstraction view of a system, the system model is composed of concurrent elements that exchange information. Therefore, in the system model the concurrency and the communication are clearly identified. The communication elements are the only way to communicate process; there is strict separation between communication and computation. This clear separation between the computation and communication simplifies the study and the analysis of the concurrent systems since formal properties of the different elements that form such systems can be obtained.

Based on separation between concurrency and communication, the HetSC methodology has been developed [28, 29]. HetSC is a methodology for the specification of concurrent heterogeneous embedded systems in SystemC. HetSC separates the computation and communication aspects of a system, creating formal executable specifications of the system. The mapping among functionality and architecture has not been done yet. Additionally, HetSC provides the mechanism for providing the system specification with the specific characteristics that a model of computation has. Therefore, HetSC enables the creation of heterogeneous system specifications where “heterogeneous” denotes that different parts of the system are described under different models of computation (MoC). The application of this methodology avoids having undesirable results such as indeterminism, deadlocks, starvation, etc. This description is performed by creating a set of channels that expresses the features of a set of MoCs.

HetSC channel	Communication Semantic
uc_inf_fifo	unlimited buffering capability
uc_fifo	limited buffering capability
uc_rv_sync	rendezvous synchronization
uc_rv_uni	rendezvous synchronization with a data transfer
uc_rv	rendezvous synchronization with bidirectional data transfer
uc_arc_seq	producer and a consumer node of a SDFG

Table 1. HetSC channels

The HetSC channels included in table 1 implement the communication semantics associated with different Untimed MoCs. These channels are explained in the section 6.

## 5. MARTE subset required for modeling MoCs

The MARTE profile is the starting point of this methodology. It has the design concepts necessary to provide the system model with the necessary formal information. This formal information consists mainly of concurrent and communications aspects. This

information is sufficient to establish the most abstract view, but this view is not enough. It is essential to add more information to the system model, in order to provide different semantics. Each different semantic establishes a specific approach and the corresponding HetSC specification is generated.

This design methodology is composed of three main elements namely concurrency elements, communication instances and encapsulate-functionality elements. The concurrency elements have the capacity to perform their own functionality concurrently, using transport-information elements to exchange data or to synchronize their execution flows among themselves. These elements are contained in encapsulated elements to provide the system with the necessary structure. The next step is to decide which elements of UML/MARTE are necessary to describe the system. The main task of this methodology is to express different communication semantics that implement the characteristics of a specific MoC. The communication semantics are understood under a specific MoC which clearly defines the relation among the different elements that compose a system. Therefore, these UML/MARTE elements have to be able to undertake this task. Additionally, other aspects of this methodology have to be covered to provide a complete design methodology.

The main MARTE subprofile that is used in this methodology is *Generic Resource Modeling* [2], whose main element is the Resource concept. It represents an entity (physical or logical) that offers services. Inheriting from this concept, this feature provides the two essential elements that form the backbone of the methodology, namely **ConcurrencyResource** and **CommunicationMedia**. A ConcurrencyResource represents an element that is capable of performing its associated flow of execution concurrently with others. The concurrency may be physical or logical. This resource represents a generic concurrent element where the hardware and software parts are not identified yet.

A CommunicationMedia is a resource that is in charge of the transport of information. It has several attributes as can be seen in figure 2. Just two of them are interesting for this methodology for now (in future work packetT is taken into account as well). *elementSize* expresses the size in bits of the elementary element that can be transmitted. *transmMode* defines the transmission mode. It could be simplex, half-duplex or full-duplex.

CommunicationMedia
element : Integer
capacity : NFP_DataTxRate
packetT : NFP_Duration
blockT : NFP_Duration
transmMode : TransmModeKind

Figure 2. CommunicationMedia attributes

In addition, this MARTE chapter provides another two elements that will be necessary to add to the CommunicationMedia of essential extra information to implement different scenarios, with different communication semantics. **StorageResource** represents memory.

Thus, it contributes adding storage capacity to the CommunicationMedia. **CommunicationEndPoint** is an element that acts as a terminal for connecting and delivering data to a CommunicationMedia. It is characterized by the *packetSize* attribute that represents the size in bits of the elementary messages. This size may or may not correspond to the media element size.

*High-Level Application Modeling* (HLAM) [2] provides the **RtService** concept. This service has real-time features and has several attributes that are shown in figure 3. *concPolicy* defines the kind of concurrency of a behavioral feature. *exeKind* denotes the execution nature property of the service. *synchKind* expresses the synchronization mechanism of the service. The latter attribute is the most interesting for this methodology. The different values of the attribute are:

- synchronous*. The client waits for the end of the invoked behaviour before continuing its own execution.
- asynchronous*. The client does not wait for the end of the invoked behaviour before continuing its own execution.
- delayedSynchronous*. The client continues to execute and will synchronize later when the invoked behaviour returns a value.
- rendezvous*. The behaviour waits for the client to start executing.
- other*. Another synchronization policy.

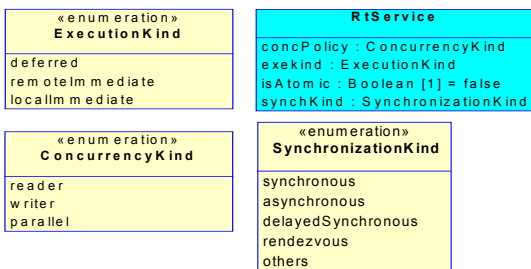





Figure 3 RtService in the HLAM subprofile

The *Generic Component Model* (GCM) [2] provides the **FlowPort** element. This type of port is used for flow-oriented communication between components and it may relay incoming, outgoing or bidirectional flows. The *direction* attribute specifies the direction of the port. The Port can be *in*, *out* and *inout*. Depending on which type of port we have, the port will be represented by a specific symbol. For an outgoing atomic FlowPort the symbol is . For an incoming atomic FlowPort the symbol is . For bidirectional atomic flow, the symbol is .

Apart from these specific MARTE elements, the concept of **Component** has been included in this methodology, as an element which encapsulates the different functionalities that may make up the system. It contributes to denote the hierarchical structure of the system.

## 5.1 Application Restriction

In this section a set of rules are introduced to restrict the application of the stereotypes described to UML metamodel elements.

A CommunicationMedia (Figure 4) is a structured class. As a rule, it has two ports that have a provided interface since only it is allowed to connect two elements for a communication instance (there is an exception to this rule). Each port is stereotyped by FlowPort. This stereotype provides the direction of the data flow. Provided interfaces stereotyped by CommunicationEndPoint, denote these interfaces act as connecting terminals of the CommunicationMedia. Each CommunicationEndPoint has a single operation. This operation represents the behaviour that concurrent elements have to call to establish the communication. This operation is stereotyped by RtService to provide the necessary characteristics to model different communication semantics.

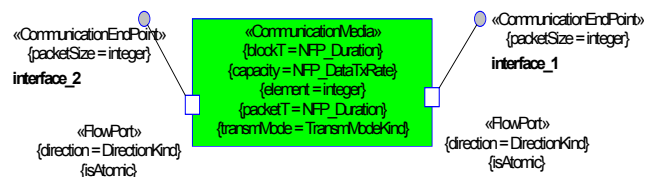


Figure 4. Structure of a CommunicationMedia

A ConcurrencyResource is modeled as a structured class (figure 5). It may be connected to a non-defined number of CommunicationMedia. For each connection, there is a port, stereotyped by FlowPort, with a required interface. This interface is connected to the corresponding CommunicationMedia that provides the necessary operation to exchange the information.

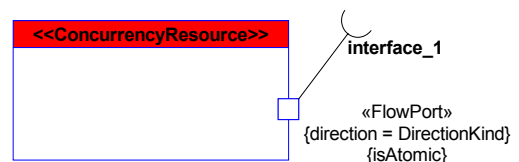


Figure 5. Structured Class of a ConcurrencyResource

As an encapsulated element, a Component is a structured class. Internally, it is formed by a set of ConcurrentResource connected by set of CommunicationMedia. Additionally, a Component may be connected to several Components thus, it has the FlowPort with the corresponding required interfaces. Additionally, a Component can have a provided interface. This case is treated in the following sections.

## 6. Using MoCs for automatic MARTE-SystemC translation

Once the MARTE subset has been presented, the next step is to explain how this subset has to be used to express the characteristics that each MoC has. To validate the model a functional execution of the description is required. From a UML/MARTE model, a process of transformation to the SystemC specification can take place since there is a clear and specific relation among the UML/MARTE modeling elements and the SystemC specification elements. This relation is explained in the following sections.

## 6.1 Methodology Foundations

As was mentioned previously, ForSyDe is the origin of this methodology. ForSyDe provides the essential concepts, such as, separation between computation and communication. This clear distinction is the “skeleton” of this design methodology.

Once the “skeleton” of the system model have been specified, the next step is to provide “identity” to this basic structure. The MARTE profile provides this “identity”. The MARTE profile has the necessary concepts to describe the concurrency and communication features at the most abstract level. Following the ForSyDe guidelines, a system should be conceived as a set of concurrent agents that are connected to exchange information. Therefore, the MARTE model should be composed of concurrent elements that are connected by communication instances that perform the data exchange. As a consequence, the basic system model is described by a set of ConcurrencyResource to denote these concurrent agents and CommunicationMedia as communication instances.

When the “identity” has been added to the “skeleton”, the next step is to enrich the system model with a set of nuances and features that provides the “personality” to the system model. The characteristics of the interaction among the concurrent agents should be understood under a certain MoC that provides this “personality”. The MoCs extend the semantic of the communication instances adding a set of characteristics that each MoC has. Therefore, it is necessary to be able to specify these MoC characteristics using MARTE. This early work attempts to specify the features of different Untimed MoCs [1].

## 6.2 Untimed MoC

In an Untimed model of computation, the time modeling is abstracted as a Causality relation [1]; the cause and the effects generated by the events that form a signal. The events communicated by the concurrent agents do not contain any timing information. The computation and the communication take an arbitrary and unknown amount of time. An order relation is denoted; the event sent first by the producer is received first by the consumer, but there is no relation among events that form different signals. Therefore, only the order of the data and the cause and effects are relevant. Kahn process networks [30], Communication Sequential processes [31] and Synchronous Data Flows [32] are the Untimed MoCs dealt with in this paper.

### 6.2.1 Kahn process network

In a Kahn process network a set of processes are communicating through the FIFO channels. The producer process writes in the channel using a non-blocking method while the consumer process needs a blocking read method. It forces the consumer to stall when no data is in the channel. There are two HetSC channels that implement the communication semantics of this MoC. **uc\_inf\_fifo** [28] is a channel with an unlimited buffering capacity. The channel **uc\_fifo** [28] has a limited buffering capacity instead. Both channels describe a unidirectional communication between two concurrent agents, where one is “the producer” and the other is “the reader”.

#### 6.2.1.1 MARTE Description

To express the buffering capacity, the StorageResource has to be added to the communication instance. As all Resources types, they inherit the *resMult* attribute. In the storage scope, *resMult*

represents how many basic storage units the communication instance has. The attribute *elementSize* of this stereotype represents the size of the element to store in bits and it should have the same value as the *elementSize* of CommunicationMedia. To express the **uc\_inf\_fifo** [28], the *resMult* attribute does not have any value. It represents a non-defining store capacity and, thus, it is understood as unlimited storage capacity. Otherwise, **uc\_fifo** [28] is represented by means of *resMult* with a specific value. To maintain consistency in the communication, in the reader side, the attribute of the *synchKind* of the RtService should be *synchronous* since the reader must have received a value to continue its associated execution flow. On the producer side, *synchKind* is *asynchronous*. Additionally, the *concPolicy* attribute should be *writer* in both RtService to express that the call to these RtService produces internal changes in the CommunicationMedia; one concurrent element writes data and the second one remove the data.

### 6.2.2 Communicating Sequential Processes

The CSP MoC is supported as a network of processes where the communication primitive is the rendezvous, that is, a concurrent element waits for the rest to reach a certain point of its execution to synchronize. When this (or these) process(es) reach such a point, then all of them can go on executing. In this MoC, it is possible to come across three different channels to cover the different situations, such as synchronization with data transfer and only synchronization:

- **uc\_rv\_sync**, this channel performs rendezvous synchronization for a set of N (with N=2 or greater) concurrent agents. The process blocks if there are less than N-1 previous arrivals of the other N-1 concurrent agents. This channel performs no kind of data transfer among concurrent agents involved in the synchronization.
- **uc\_rv\_uni**, this channel performs rendezvous synchronization between two concurrent agents with a unidirectional data transfer.
- **uc\_rv**, this channel performs rendezvous synchronization between two concurrent agents with a bidirectional data transfer. It is the only HetSC channel that implements this kind of communication.

#### 6.2.2.1 MARTE Description

The main characteristic of CSP MoC is the synchronization among the execution flows of the processes. Synchronization is denoted by the values *delayedSynchronous* and *rendezvous* of the *synchKind* attribute. In this early work only the structure of the system is expressed. These two attribute values are essential to specify whether the concurrency elements involved in the communication only perform synchronization or synchronization with data transfer. However, to avoid inconsistent calls to the real-time services, the behaviour of the concurrent elements is described by means of an activity diagram, adding the concepts provided by the Time Modelling subprofile of MARTE. Additionally, a set of guidelines on how to use all these concepts should be provided. This work is on going and thus, it is not dealt with in this paper. In the context of this paper, *delayedSynchronous* and *rendezvous* attribute values provide the behaviour semantic that fits with the communication semantic of the corresponding HetSC channels. To express the communication semantics of the **uc\_rv\_sync** channel [28] (only

synchronization), *rendezvous* attribute should be the value in all RtService. This is the case where several concurrent elements can be connected to the same communication instance. The *concPolicy* attribute should be *reader* in both RtService.

When the synchronization is accompanied by the data transfer, there are two scenarios, unidirectional communication and bidirectional communication between two concurrency elements. In the case of unidirectional communication, *uc\_rv\_uni* [28], there are two different roles, “producer” and “consumer”. *synchKind* attribute of the “consumer” RtService has to be *delayedSynchronous*, the client continues to execute and will synchronize later when the invoked behaviour returns a value. Again, a definition of guidelines is necessary to prevent inconsistency in the use of this attribute. It is future work. In the “producer” RtService, *synchKind* attribute is *rendezvous* to denote synchronization. The *concPolicy* attribute should be *writer* in both RtService.

The main characteristic of the *uc\_rv* channel is that it implements bidirectional communication. To express this communication semantic, the *transmMode* attribute of the CommunicationMedia is essential. *transmMode* has three possible values. In all cases where a unidirectional communication is established, the correct value of the *transmMode* attribute is *simplex*, which represents one-way communication of data. In this case, the FlowPorts of the communication instance are *in* for the producer side and for the reader side the Flowport is *out*. Bidirectional communication may be expressed by means of two values, *half-duplex* and *full-duplex*. These values describe two different communication semantics; *full-duplex* allows simultaneity in the communication and *half-duplex* does not. The value that fits with the communication semantics of the *uc\_rv* is *full-duplex* since this HetSC channel allows a simultaneous communication. The corresponding Flowports are *inout*. The *synchKind* attribute of both RtService of the communication instance should be *delayedSynchronous* since this attribute value includes synchronization and a data transmission, which means the elements involved in the communication receive their expected data at a specific point in their execution flows. Again, the *concPolicy* attribute should be *writer* in both RtService.

### 6.2.3 Synchronous Data Flow

SDF MoC is represented as a network of computation nodes communicated by arcs and with unidirectional communication (called SDF graph). Each arc comes only from one node and arrives only to one node. Each arc has two associated rates, a production and a consumption rate. The consumption node is triggered only when there are enough data in the input arcs, denoted by the consumption rate. In addition, it expresses the number of data consumed in each computation. Each input arc has enough tokens whenever it has at least a number of tokens available greater or equal to the input arc consumption rate. The tokens are processed and output arcs are written according to the production rates. The *uc\_arc\_seq* [28] is the HetSC channel that implements this communication semantics.

#### 6.2.3.1 MARTE Description

To express the communication semantics of the SDF, it is necessary to model both consumption rate and production rate. Here, the StorageResource stereotype plays an essential role. This stereotype is associated with the communication media to denote

the store capacity. However, in this MoC, this store capacity is understood to be the number of data that have to be stored in order to trigger the computation of the consumption node. The *resMult* of the StorageResource attribute specifies the consumer rate. The accumulated data are grouped in a packet. The size of this packet is denoted by the *packetSize* attribute of the CommunicationEndPoint of the “reader” side. On the other side, the producer rate is denoted by the *packetSize* attribute of its corresponding CommunicationEndPoint. The value of this attribute is a multiple of the *elementSize* attribute of the CommunicationMedia (and the StorageResource). This multiple is the producer rate. *synchKind* attribute is *synchronous* on the consumer side and *asynchronous* on the producer side. The *concPolicy* attribute should be *writer* in both RtService.

## 6.3 Translation to SystemC elements

Apart from the translation from a CommunicationMedia with a specific semantic to the corresponding HetSC channel, there are several elements of this methodology that have a straightforward transformation into SystemC elements as is shown in the table 2.

UML/MARTE	SystemC
ConcurrencyResource	sc_thread
Component	sc_module
FlowPort	sc_port, sc_export

Table 2

In the HetSC methodology, the computation is mapped onto *sc\_thread*, which is one of the concurrent units of SystemC. Therefore, ConcurrencyResource is transformed into *sc\_thread*. *sc\_module* is the SystemC element used to establish the hierarchy of the system. It is the way to identify independently functioning parts of the system. This is done by the UML Component in this methodology. All the elements have Flowports to establish the communication. Only the FlowPorts that are part of a Component are transformed into SystemC ports. Depending on the situation, a FlowPort is a *sc\_port* or a *sc\_export*. As a rule, all the Flowports of a Component corresponds to a *sc\_port*, except in one case shown in Figure 6.

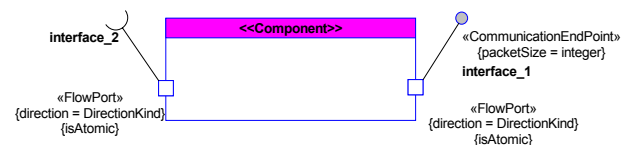


Figure 6. Usage of Component

Figure 6 shows a Component with a provided interface. This means that internally, the Component has a CommunicationMedia where only one provided interface of this communication instance is connected to an internal component element. The other interface implemented by this CommunicationMedia is offered by the Component to its environment. This situation enables the connection of a system element directly to the Component.

## 7. Conclusions

This paper proposes the use of MARTE to capture concurrency and communication in complex embedded systems. The proposed solution could enable formal descriptions based on ForSyDe to be obtained (this work is currently being done). A subset of UML/MARTE elements has been selected to cover two essential features, computation and communication. In addition, this paper explores the capabilities of the MARTE profile to create system models with specific, different communication semantics. These different communication semantics are understood under different untimed MoCs and are described through enriching the communications instances with additional communications features provided by another subset of MARTE. With these new features, the communication instances have the necessary information to be transformed into the corresponding HetSC channel. Moreover, additional UML/MARTE elements are used to describe the structure of the system. These elements have a straightforward translation to SystemC elements.

## 8. Future work

The work presented in this paper is still on-going. The aim is to provide the designer with a complete methodology to create models under different models of computation. The first approximation has been completed, but it is not enough. It is necessary to enrich the model with additional behavior semantics. To achieve this goal, the Time Modeling sub-profile [26][27] of MARTE will be used. The functionality of the ConcurrentResource will be expressed by means of Activity diagrams. However, these diagrams should be created under certain guidelines to respect the specific MoC semantics and so avoid possible inconsistency in the calls of the RtServices. Combining Time Modeling facilities and activity diagrams, the methodology will be complete and capable of expressing different MoCs. In addition, this paper has dealt with untimed MoC, but it will be extended to cover Synchronous MoCs. These MoCs have their corresponding HetSC channels to implement different communication semantics. The exploration of these new requirements will establish whether new MARTE elements are necessary.

## 9. Acknowledgements

Work supported by ICT project SATURN (FP7-216807).

## 10. References

- [1] Jantsch A. 2004. *Modeling Embedded Systems and SoCs*. Morgan Kaufmann, Elsevier Science.
- [2] <http://www.imit.kth.se/info/FOFU/ForSyDe/>
- [3] Object Management Group (2008) UML Profile for MARTE, beta 2. OMG document number: ptc/08-07-09.
- [4] F.Herrera and E.Villar A Framework for Embedded System Specification under Different Models of Computation in SystemC, Annual ACM IEEE Design Automation Conference Proceedings of the 43rd annual conference on Design automation. 2006
- [5] The open SystemC Initiative [www.systemc.org](http://www.systemc.org).
- [6] [www.omg.org/spec/FUML/](http://www.omg.org/spec/FUML/)
- [7] SysML Partners web site. [www.sysml.org/](http://www.sysml.org/)
- [8] M. Pauwels, Y. Vanderperren, G. Sonck, P. van Oostende, W. Dehaene, T. Moore. A Design Methodology for the Development of a Complex System-On-Chip using UML and Executable System Models. FDL'02.
- [9] S. Bocchio, E. Riccobene, A. Rosti, P. Scandurra. A SoC design flow based on UML 2.0 and SystemC. In DAC, Workshop UML-Sock'05.
- [10] E. Riccobene, P. Scandurra, A. Rosti and S. Bocchio. A UML 2.0 Profile for SystemC. STMicroelectronics Technical Report, 2004.
- [11] S. Bocchio, E. Riccobene, A. Rosti and P. Scandurra. An Enhanced SystemC UML Profile for Modeling at Transaction-Level. Embedded Systems Specification and Design Languages Selected contributions from FDL'07.
- [12] W. Mueller, A. Rosti, S. Bocchio, E. Riccobene, P. Scandurra, W. Dehaene, Y. Vanderperren. UML for ESL Design – Basic Principals, Tools, and Applications. IEEE/ACM 2006.
- [13] Y. Vanderperren, W. Mueller, W. Dehaene. UML for electronic systems design: a comprehensive overview. Journal on Design Automation for Embedded Systems, Springer Verlag, August 2008.
- [14] C. Mraidha, Y. Tanguy, C. Jouvray, F. Terrier, S. Gérard. An Execution Framework for MARTE-based Models. Thirteenth IEEE International Conference on the Engineering of Complex Computer Systems, ICECCS 2008
- [15] S. Gérard, "Modélisation UML exécutable pour les systèmes embarqués de l'automobile," Evry Université, 2000.
- [16] S. Gérard, F. Terrier, and Y. Tanguy, "Using the model paradigm for Real-Time Systems Development," presented at OOIS'02-MDSD 2002.
- [17] É.Piel, R. Ben Atitallah, P. Marquet, S. Meftali, S. Niar, A. Etien, Jean-Luc Dekeyser, P. Boulet. Gaspard2: from MARTE to SystemC Simulation. DATE'08.
- [18] M. Mura, A. Panda, M. Prevostini. Executable Models and Verification from MARTE and SysML: a Comparative Study of Code Generation Capabilities. DATE'08.
- [19] C. André, F. Mallet, R. de Simone. Modeling Time(s). MoDELS 2007.
- [20] P.Andersson, M.Höst. UML and SystemC a Comparison and Mapping Rules for Automatic Code Generation. Embedded Systems Specification and Design Languages Selected contributions from FDL'07.
- [21] J.Kreku, M.Hoppari, T.Kestilä, Y.Qu, J-P.Soininen, P.Andersson, K.Tiensyrjä. Combining UML2 application and SystemC platform modelling for performance evaluation of real-time embedded systems. EURASIP Journal on Embedded Systems Volume 2008.
- [22] J.Kreku, M.Hoppari, T.Kestilä. SystemC workload model generation from UML for performance simulation. FDL '07.
- [23] W.Raslan, A.Sameh. Mapping SysML to SystemC. FDL '07.
- [24] W.Raslan, A.Sameh. System-Level Modeling and Design Using SysML and SystemC. DAC 2007.

- [25] F. Bruschi, E. Di Nitto, D. Sciuto. SystemC Code Generation from UML Models. Forum on Specification and Design Languages '02.
- [26] C. André, F. Mallet, R. de Simone. Modeling Time(s). In Proceedings of the ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MoDELS/UML), October 2007, TN, USA. Springer Verlag, LNCS 4735, pp. 559--573.
- [27] C. André, F. Mallet, R. de Simone. Time modeling in MARTE. FDL 2007.
- [28] <http://www.teisa.unican.es/HetSC/>
- [29] F. Herrera, E. Villar : "A Framework for Heterogeneous Specification and Design of Electronic Embedded Systems in SystemC", ACM Transactions on Design Automation of Electronic Systems, Special Issue on Demonstrable Software Systems and Hardware Platforms, V.12, Issue 3, N.22. 2007.
- [30] G. Kahn. The semantics of a simple language for parallel programming. In Proceedings of the International Federation for Information Processing Working Conference on Data Semantics. 1974.
- [31] C. A. R. Hoare. Communicating sequential processes. Commun. ACM 21, 8. 1978.
- [32] E. A. Lee, D. G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. IEEE Trans. Comput. 36, 1. 1987.
- [33] S. Taha, A. Radermacher, S. Gerard Jean-Luc Dekeyser. MARTE: UML-based Hardware Design from Modeling to Simulation. FDL'2007, Barcelona, Spain, September 2007.
- [34] S. Revol, S. Taha, F. Terrier, A. Clouard, S. Gerard, A. Radermacher, Jean-Luc Dekeyser. Unifying HW Analysis and SoC Design Flows by Bridging Two Key Standards: UML and IP-XACT.