# 11

# Formal Foundations for the Generation of Heterogeneous Executable Specifications in SystemC from UML/MARTE Models

Pablo Peñil, Fernando Herrera and Eugenio Villar
*Microelectronics Engineering Group of the University of Cantabria*
*Spain*

## 1. Introduction

Technological evolution is provoking an increase in the complexity of embedded systems derived from the capacity to implement a growing number of elements in a single, multi-processing, system-on-chip (MPSoC).

Embedded system heterogeneity leads to the need to understand the system as an aggregation of components in which different behavioural semantics should cohabit. Heterogeneity has two dimensions. On the one hand, during the design process, different execution semantics, specifically in terms of time (untimed, synchronous, timed) can be required in order to provide specific behaviour characteristics for the concurrent system elements. On the other hand, different system components may require different models of computation (MoCs) in order to better capture their functionality, such as Kahn Process Networks (KPN), Synchronous Reactive (SR), Communicating Sequential Processes (CSP), TLM, Discrete Event (DE), etc.

Another aspect affecting the complexity of current embedded systems derives from their structural concurrency. The system should be conceived as an understandable architecture of cooperating, concurrent processes. The cooperation among these concurrent processes is implemented through information exchange and synchronization mechanisms. Therefore, it is essential to deal with the massive concurrency and parallelism found in current embedded systems and provide adequate mechanisms to specify and verify the system functionality, taking into account the effects of the different architectural mappings to the platform resources.

In this context, the challenge of designing embedded systems is being dealt with by application of methodologies based on Model Driven Architecture (MDA) (MDA guide, 2003). MDA is a developing framework that enables the description of systems by means of models at different abstraction levels. MDA separates the specification of the system's generic characteristics from the details of the platform where the system will be implemented. Specifically, in Platform Independent Models (PIMs), designers capture the relevant properties that characterize the system; the internal structure, the communication mechanisms, the behavior of the different components, etc. Therefore, PIMs provide a general, synthetic representation that is independent and, thus, decoupled from the final

system implementation. High-level PIM models are the starting point of ESL methodologies, and they are crucial for fast validation and Design Space Exploration (DSE). PIMs can be implemented on different platforms leading to different Platform Specific Models (PSMs). PSMs enable the analysis of performance characteristics of the system implementation.

The most widely accepted and used language for MDA is the Unified Modelling Language (UML) (UML, 2010). UML is a standard graphical language to visualize, specify and document the system. From the first application as object-oriented software system modelling, the application domain of UML has been extended. Nowadays, UML is used to deal with electronic system design (Lavagno et al. 2003). Nevertheless, UML lacks the specific semantics required to support embedded system specification, modelling and design. This lack of expressivity is dealt with by means of specific profiles that provide the UML elements with the necessary, precise semantics to apply the UML modelling capabilities to the corresponding domain.

Specifically in the embedded system domain, UML should be able to deal with design aspects such as specification, analysis, architectural mapping and implementation of complex, HW/SW embedded systems. The MARTE UML profile (UML Profile for MARTE, 2009), which was created recently, was developed in order to model and analyze real-time embedded systems, providing the concepts needed to describe real-time features that specify the semantics of this kind of systems at different abstraction levels. The MARTE profile has the necessary concepts to create models of embedded systems and provide the capabilities that enable the analysis of different aspects of the behaviour of such systems in the same framework. By using this UML profile, designers will be able to specify the system both as a generic entity, capturing the high-level system characteristics and, after a refinement process, as a detailed architecture of heterogeneous components. In this way, designers will be assisted by design flows with a generic system model as an initial stage. Then, by means of a refinement process supported by modelling and analysis tools, they will be able to decide on the most appropriate architectural mapping.

As with any UML profile, MARTE is not associated with any explicit execution semantics. As a consequence, no executable model can be directly extracted for simulation, functional verification and performance estimation purposes. In order to address this need, SystemC (Open SystemC) has been proposed as the specification and simulation framework for MARTE models. From the MARTE model, an executable model in SystemC can be inferred establishing a MARTE/SystemC relationship.

The MARTE/SystemC relationship is established in a formal way. The corresponding formalism should be as general as possible in order to enable the integration of heterogeneous components interacting in a predictable and well-understood way (horizontal heterogeneity) and to support the vertical heterogeneity, that is, refinement of the model from one abstraction level to another. Finally, this formalism should remove the ambiguity in the execution semantics of the models in order to provide a basis for supporting methodologies that tackle embedded system design.

For this purpose, the ForSyDe (Formal System Design) meta-model (Jantsch, 2004) was introduced. ForSyDe was developed to support the design of heterogeneous embedded systems by means of a formal notation. ForSyDe enables the production of a formal specification that captures the functionality of the system as a high abstraction-level model.

From these initial formal specifications, a set of transformations can be applied to refine the model into the final system model. This refinement process generally involves MoC transformation.

A system-level modelling and specification methodology based on UML/MARTE is proposed. A subset of UML and MARTE elements is selected in order to provide a generic model of the system. This subset of UML/MARTE elements is focused on capturing the generic concurrency and the communication aspects among concurrent elements. Here, system-level refers to a PIM able to capture the system structure and functionality independently of its final implementation on the different platform resources. The internal system structure is modelled by means of Composite Structure diagrams. MARTE *concurrency resources* are used to model the concurrent processes composing the concurrent structure of the system. The communication elements among the concurrent processes are modelled using the CommunicationMedia stereotype. The concurrent processes and the communication media compose the Concurrent&Communication (C&C) structure of the system. The explicit identification of the concurrent elements facilitates the allocation of the system application to platforms with multiple processing elements in later design phases.

In order to avoid any restrictions on the designer, the methodology does not impose any specific functionality modelling of concurrent processes. Nevertheless, with no loss of generality, UML activity diagrams are used as a meta-model of functionality. The activity diagram will provide formal support to the C&C structure of the system, explaining when each concurrent process takes input values, how it computes them and when the corresponding outputs are delivered.

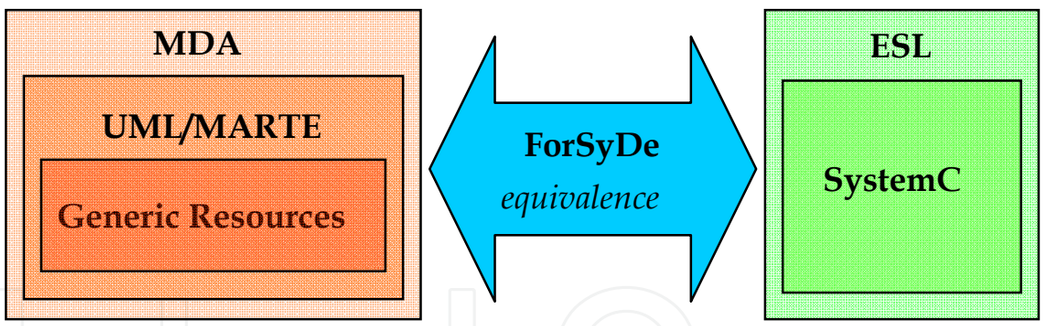



Fig. 1. ForSyDe formal link between MDA and ESL.

Based on the MARTE/SystemC formal link supported by ForSyDe, the methodology enables untimed SystemC executable specifications to be obtained from UML/MARTE models. The untimed SystemC executable specification allows the simulation, validation and analysis of the corresponding UML/MARTE model based on a clear simulation semantics provided by the underlying formal model. Although the formal model could be kept transparent to the user, the model defines clear simulation semantics associated with the MARTE model and its implementation in the SystemC model, which can be fully understood by any designer. Therefore, the ForSyDe meta-model formally supports interoperability between MARTE and SystemC.

In this way, the gap between MDA and ESL is formally bridged by means of a conceptual mapping. The mapping established among UML/MARTE and SystemC will provide

consistency in order to ensure that the SystemC executable specification obtained is equivalent to the original UML/MARTE model. The formal link provided by ForSyDe enables the abstract executive semantics of both the UML/MARTE model and its corresponding SystemC executable specification to be reflected (Figure 4.). This demonstrates the equivalence among the two design flow stages, provides the required consistency to the mapping established between the two languages and ensures that the transformation process is correct-by-construction.

## 2. Related work

Several works have shown the advantages of using the MARTE profile for embedded system design. For instance, in (Taha et al, 2007) a methodology for modelling hardware by using the MARTE profile is proposed. In (Vidal et al, 2009), a co-design methodology for high-quality real-time embedded system design from MARTE is presented.

Several research lines have tackled the problem of providing an executive semantics for UML. In this context, two main approaches for generating SystemC executable specifications from UML can be distinguished. One research line is to create a SystemC profile in order to capture the semantics of SystemC facilities in UML diagrams (Bocchio et al., 2008). In this case, SystemC is used both as modelling and action language, while UML enables a graphical capture. A second research line for relating UML and SystemC consists in establishing mapping rules between the UML metamodel and the SystemC constructs. In this case, pure UML is used for system modelling, while the SystemC model generated is used as the action language. Mapping rules enable automatic generation of the executable SystemC code (Andersson & Höst, 2008). In (Kreku et al., 2007) a mapping between UML application models and the SystemC platform models is proposed in order to define transformation rules to enable semi-automatic code generation.

A few works have focused on obtaining SystemC executable models from MARTE. Gaspard2 (Piel et al. 2008) is a design environment for data-intensive applications which enables MARTE description of both the application and the hardware platform, including MPSoC and regular structures. Through model transformations, Gaspard2 is able to generate an executable TLM SystemC platform at the timed programmers view (PVT) level. Therefore, Gaspard2 enables flows starting from the MARTE post-partitioning models, and the generation of their corresponding post-partitioning SystemC executables.

Several works have confronted the challenge of providing a formal basis for UML and SystemC-based methodologies. Regarding UML formalization, most of the effort has been focused on providing an understanding of the different UML diagrams under a particular formalism. In (Störrle & Hausmann, 2005) activity diagrams are understood through the Petri net formalism. In (Eshuis & Wieringa, 2001) formal execution semantics for the activity diagrams is defined to support the execution workflow. In the context of MARTE, the Clock Constraint Specification Language (CCSL) (Mallet, 2008) is a formalism developed for capturing timing information from MARTE models. However, further formalization effort is still required.

A significant formalization effort has also been made in the SystemC context. The need to conceive the whole system in a model has brought about the formalization of abstract and heterogeneous specifications in SystemC. In (Kroening & Sharygna, 2005) SystemC

specifications including software and hardware domains are formalized to support verification. In (Maraninchi et al., 2005) TLM descriptions are related to synchronous systems are formalized. In (Traulsem et al., 2007) TLM descriptions related to asynchronous systems are formalized. Comprehensive untimed SystemC specification frameworks have been proposed, such as SysteMoC (Falk et al., 2006) and HetSC (Herrera & Villar 2006). These methodologies take advantage of the formal properties of the specific MoCs they support but do not provide formal support for untimed SystemC specifications in general. Previous work on the formalization of SystemC was focused on simulation semantics. These approaches were inspired by previous formalization work carried out for hardware design languages such as VHDL and Verilog. In (Mueller et al., 2001), SystemC processes were seen as distributed abstract state machines which consume and produce data in each delta cycle. In this way the corresponding model is strongly related to the simulation semantics. In (Salem, 2003), denotation semantics was provided for the synchronous domain. Efforts towards more abstract levels address the formalization of TLM specifications. In (Ecker et al., 2006), SystemC specifications including software and hardware functions are formalized. In (Moy et al., 2008) TLM descriptions are related to synchronous and asynchronous formalisms.

Nevertheless, a formal framework for UML/MARTE-SystemC mapping based on common formal models of both languages is required. A good candidate to provide this formal framework is the ForSyDe metamodel (Janstch, 2004). The Formal System Design (ForSyDe) formalism is able to provide a synthetic notation and understanding of concurrent and heterogeneous specifications. ForSyDe covers modelling of time at different abstraction levels, such as untimed, synchronous and timed. Moreover, ForSyDe supports verification and transformational design (Raudvere et al. 2008).

## 3. ForSyDe

ForSyDe provides the mechanism to enable a formal description of a system. ForSyDe is mainly focused on understanding concurrency and time in a formal way representing a system as a concurrent model, where processes communicate through signals. In this way, ForSyDe provides the foundations for the formalization of the C&C structure of the system. Furthermore, ForSyDe formally supports the functionality descriptions associated with each concurrent process.

Processes and signals are metamodelling concepts with a precise and unambiguous mathematical definition. A ForSyDe signal is a sequence of events where each event has a tag and a value. The tag is often given implicitly as the position in the signal and it is used to denote the partial order of events. In ForSyDe, processes have to be seen as mathematical relations among signals. The processes are concurrent elements with an internal state machine. The relation among processes and signals is shown in Figure 2.

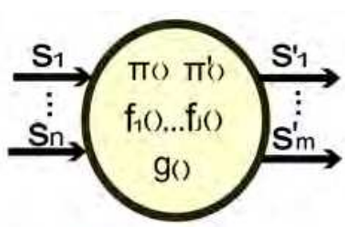



Fig. 2. ForSyDe metamodel representation.

From a general point of view; a ForSyDe process p is characterized by the expression:

$$p(s_1 ... s_n) = s'_1 ... s'_m \tag{1}$$

The process p takes a set of signals ($s_1 \ldots s_n$) as inputs and produces a set of outputs ($s'_1 \ldots s'_m$), where $\forall\ 1 \le i \le n \wedge 1 \le j \le m$ with $n, m \in \mathbb{N}$; $s_i, s_j \in S$ where $s_k$ are individual signals and S is the set of all ForSyDe signals.

ForSyDe distinguishes three kinds of signals namely untimed signals, synchronous signals and timed signals. Each kind of MoC is determined by a set of characteristics which define it. Based on these generic characteristics, it is possible to define a particular MoC's specific semantics.

Expressions (2) and (4) denote an important, relevant aspect that characterizes the ForSyDe processes, the data consumed/produced.

$$\begin{aligned} &\pi(\nu_1, s_1) = \langle a_1(z) \rangle \\ &... \\ &\pi(\nu_n, s_n) = \langle a_n(z) \rangle \end{aligned} \tag{2}$$

$$\begin{aligned} &with \\ &\nu_n(z) = \gamma(\omega_q) \end{aligned} \tag{3}$$

$$\begin{aligned} &\pi(\nu_1', \hat{s}_1') = \langle a'_1(z) \rangle \\ &... \\ &\pi(\nu_m', \hat{s}_m') = \langle a'_m(z) \rangle \end{aligned} \tag{4}$$

$$\begin{aligned} &with \\ &\nu_m'(z) = length(a'_m(z)) \end{aligned} \tag{5}$$

A partition π(ν,s) of a signal *s* defines an ordered set of signals ⟨$a_n$⟩ that "almost" forms the original signal *s*. The brackets ⟨...⟩ denote a set of ordered elements (events or signals). The function ν(z) defines the length of the subsignal $a_n(z)$; the semantics associated with the ν(z) function is: $\nu_n(0)$ = length($a_n(0)$); $\nu_n(1)$ = length($a_n(1)$) ... where z denotes the number of the data partition.

For the input signals, the length of these subsignals depends on which state the process is, denoted by the expression (3), where γ is the function that determines the number of events consumed in this state. The internal state of the process is denoted by $\omega_q$ with $q \in \mathbb{N}_0$. In some cases, $\nu_n(z)$ does not depend on the process state and thus $\nu_n(z)$ is a constant, denoted by the expression ν(z) = c with $c \in \mathbb{N}$.

For the output signals, the length is denoted by expression (5). The output subsignals $a'_1 \ldots a'_m$ are determined by the corresponding output function $f_\alpha$ that depends on the input subsignals $a_1 \ldots a_n$ and the internal state of the process $\omega_q$, expression (6).

$$f_{\alpha}((a_1 \ldots a_n), \omega_q) = (a'_1 \ldots a'_m) \tag{6}$$

where $\forall\ 1 \leq \alpha \leq j \wedge j \in \mathbb{N}$

The next internal state of the process is calculated using the function g:

$$g((a_1 \ldots a_n), \omega_q) = \omega_{q+1} \tag{7}$$

where $\forall\ 1 \leq i \leq n \wedge n \in \mathbb{N}_0$, $a_i \in S$, $\forall\ q \in \mathbb{N}_0$, $\omega_q \in E$. E is the set of all events, that is, untimed events, synchronous events and timed events respectively.

ForSyDe processes can be characterized by the four tuple TYPEs ⟨TI, TO, NI, NO⟩. TI and TO are the sets of signal types for the input and output signals respectively. The signal type is specified by the value type of its corresponding events that made up the signal. NI = $\{v_1(i) \ldots v_n(i)\}$ is the set of partitioning functions for the n input signals; NO=$\{v_1'(i) \ldots v_n'(i)\}$ is the set of partitioning functions of the m output signals.

The advance of time in ForSyDe processes is understood as a totally ordered sequence of evaluation cycles. In each evaluation cycle (ec) "a process consumes inputs, computes its new internal state, and emits outputs" (Jantsch, 2004). After receiving the inputs, the process reacts and then, it computes the outputs depending on its inputs and the process's internal state.

## 4. AVD system

In order to illustrate the formal foundations between UML/MARTE and SystemC a video decoder is used, specifically an Adaptive Video decoder (AVD) system. Adaptive software is a new paradigm in software programming which addresses the need to make the software more effective and thus reusable for new purposes or situations it was not originally designed for. Moreover, adaptive software has to deal with a changing environment and changing goals without the chance of rewriting and recompiling the program. Therefore, dynamic adaptation is required for these systems. Adaptive software requires the representation of the set of alternative actions that can be taken, the goals that the program is trying to achieve and the way in which the program automatically manages change, including the way the information from the environment and from the system itself is taken.

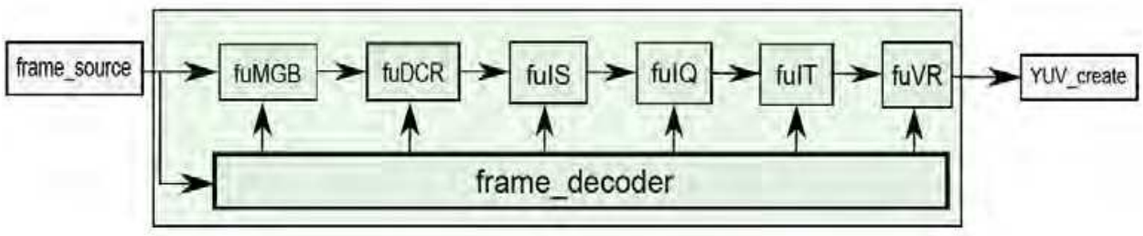



Fig. 3. Block diagram of the Adaptive Video decoder.

Specifically, the AVD specification is based on the RVC decoder architecture (Jang et al., 2008). Figure 3 illustrates a simplified scheme of the AVD architecture. The RVC architecture

divides the decoder functionality into a set of functional units (fu). Each of these functional units is in charge of a specific video decoding functionality. The *frame_decoder* functional unit is in charge of parsing and decoding the incoming MPEG frame. This functional unit is enabled to parse and extract the forward coding information associated with every frame of the input video stream. The coding information is provided to the functional units *fuIS* and *fuIQ*. The macroblock generator (*fuMGB*) is in charge of structuring the frame information into macroblocks (where a macroblock is a basic video information unit, composed of a group of blocks). The inverse scan functional unit (*fuIS*) implements the Inverse zig-zag scan. The normal process converts a matrix of any size into a one-dimensional array by implementing the zig-zag scan procedure. The inverse function takes in a one-dimensional array and by specifying the desired number of rows and columns, it returns a matrix having the specified dimensions. The inverse scan constructs an array of 8x8 DCT coefficients from a one-dimensional sequence. The *fuIQ* functional unit performs the Inverse Quantization. This functional unit implements a parameter-based adaptive process. The *fuIT* functional unit can perform the Inverse Transformation by applying an inverse DCT algorithm (IDCT), or an inverse Haar algorithm (IHAAR). Finally, the *fuVR* functional unit is in charge of video reconstruction.

The *frame _source* and the *YUV_create* blocks make up the environment of the AVD system. The *frame_source* block provides the frames of a video file that the AVD system decodes later. The *YUV_create* block rebuilds the video (in a .YUV video file) and checks the results obtained.

### 4.1 UML/MARTE model from the AVD system

The system is designed as a concurrent entity; the functionality of each functional unit is implemented by concurrent elements. Each one of these concurrent elements is allocated to an UML component and identified by the MARTE stereotype <<ConcurrencyResource>>. This MARTE generic resource models the elements that are capable of performing its associated execution flow concurrently with others. *Concurrency resources* enable the functional specification of the system as a set of concurrent processes. The information is transmitted among the *concurrent resources* by means of communicating elements identified by the MARTE stereotype <<CommunicationMedia>>. Both *ConcurrencyResource* and *CommunicationMedia* are included in MARTE subprofile Generic Resource Modelling (GRM). This gives the designer complete freedom in deciding on the most appropriate mapping of the different functional components of the system specification to the available executing resources. These MARTE elements are generic in the sense that they do not assume a specific platform mapping to HW or to SW. Thus, they are suitable for system-level pre-partition modelling.

Depending on the parameters defining the *communication media*, several types of channels can be identified. Based on the type of channels used, several MoCs can be identified (Peñil et al, 2009). When a specific MoC is found, the design methodologies associated with it can be used taking advantage of the properties that that MoC provides. Additional kinds of channels can be identified, the border channels. A border channel is a communication media that enables the connections of different MoC domains, which have their own properties and characteristics. The basic principle of the border channel semantics is that from each MoC side, the border channel is seen as the channel associated with the MoC. In the case of

*channel_4* of Figure 4, this communication media establishes the connection among the KPN MoC domains (Kanh,1974) and the CSP MoC domains (Hoare, 1978). This border channel is inferred from a *communication media* with a storage capacity provided by the stereotype <<StorageResource>>. In order to capture the unlimited storage capacity that characterizes the KPN channels, the tag *resMult* should not be defined. The communication is carried by the calls to a set of methods that a *communication media* provides. These methods are MARTE <<RtService>>. The *RtService* associated with the KPN side should be *asynchronous* and *writer*. In the CSP side, the *RtService* should be d*elayedSynchronous*. This attribute value expresses synchronization with the invoked service when the invoked service returns a value. In this *RtService* the value of *concPolicy* should be *writer* so that the data received from the *communication media* in the synchronization is consumed and, thus, producing side effects in the *communication media*. The *RtServices* are the methods that should be called by the *concurrency resources* in order to obtain/transmit the information.

Another communication (and interaction) mechanisms used for communicating threads is performed through protected shared objects. The most simple is the shared variable. A shared variable is inferred from a *communication media* that requires storage capacity provided by the MARTE stereotype <<StorageResource>>. Shared variables use the same memory block to store the value of a variable. In order to model this memory block, the tag *resMult* of the *StorageResource* stereotype should be one. The communication media accesses that enable the writings are performed using *Flowport* typed as *in*. A *RtService* is provided by this F*lowPort* and this *RtService* is specified as *asynchronous* and as *writer* in the tags *synchKind* and *concPolicy* respectively. The tag value *writer* expresses that a call to this method produces side effects in the *communication media*, that is, the stored data is modified in each writing access. Regarding the reading accesses, they are performed through *out* flow ports. The value of the *synchKind* should be *synchronous* to denote that the corresponding *concurrency resource* waits until receiving the data that should be delivered by the *communication media*. The value of *concPolicy* should be *reader* to denote that the stored data is not modified and, thus, several readings of the same data are enabled.

Figure 4 shows a sketch of a complete UML/MARTE PIM that describes the AVD system. Figure 4 is focused on the *MGB* component showing the components that are connected to the *MGB* component and the channels used for the exchange of information between this component and its specific environment. Based on this AVD component, a complete example of the ForSyDe interrelation between UML/MARTE and SystemC will be presented. However, before introducing this example, it is necessary to describe the ForSyDe formalization of the subset of UML/MARTE elements selected. For that purpose, the *IS* component is used.

### 4.2 Computation & communication structure

The formalization is done by providing a semantically equivalent ForSyDe model of the UML/MARTE PIM. Such a model guarantees the determinism of the specification and enables the application of the formal verification and refinement methodologies associated with ForSyDe. As was mentioned before, the ForSyDe metamodel is focused on the formal understanding of the communication and processing structure of a system and the timing semantics associated with each processing element's behaviour. Therefore, in order to obtain a ForSyDe model, all the system information associated with an UML/MARTE model

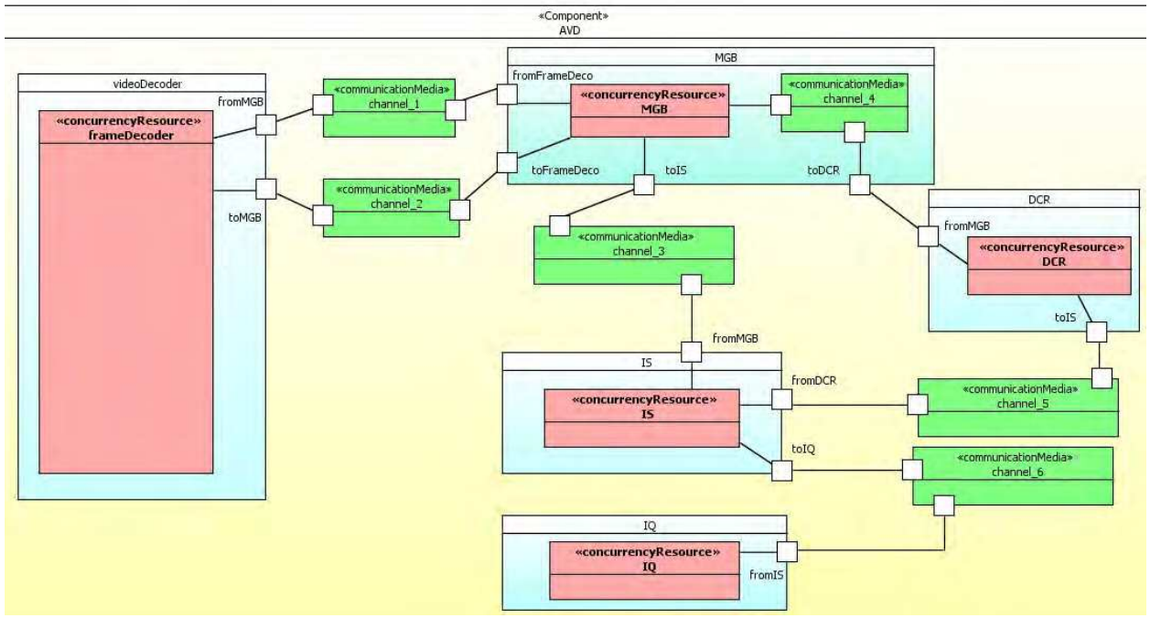



Fig. 4. Sketch of the UML/MARTE model that describes the AVD system.

related to the system structure has to be ignored. All the model elements that determine the hierarchy system structure such as UML components, UML ports, etc. have to be removed. In this way, the resulting abstraction is a model composed of the processing elements (*concurrency resources*) and the communicating elements (*communication media*). This C&C model determines the abstract semantics associated with the model and, by extension, determines the system execution semantics. Figure 5 shows the C&C abstraction of Figure 4 where only the *concurrency resources* and the *communication media* are presented.

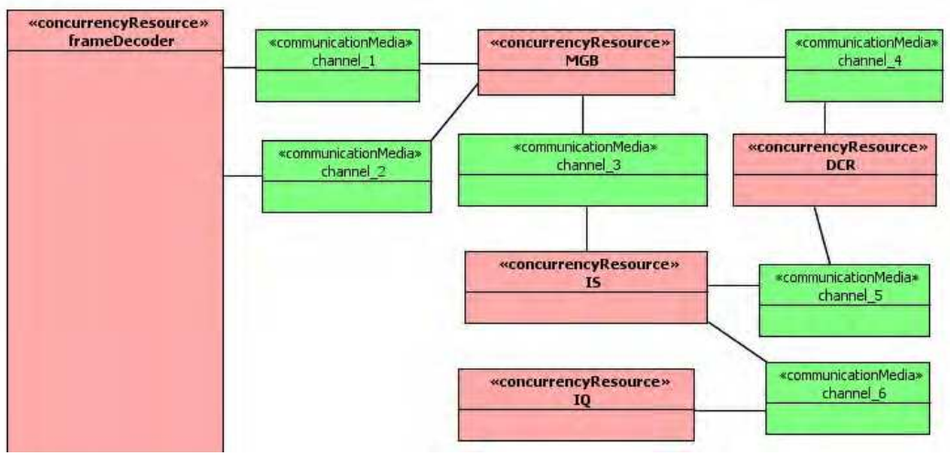



Fig. 5. C&C abstraction of the model in Figure 4.

### 4.3 ForSyDe representation of C&C structure

While the extraction of the C&C model is maintained in the UML/MARTE domain, the second step of the formalization consists in the abstraction of this UML/MARTE C&C

model as the semantically equivalent ForSyDe model. More specifically, the ForSyDe abstraction means the specification from the UML/MARTE C&C model of the corresponding processes and signals; the timing abstraction (untimed, synchronous, etc); the input and output partitions; and the specific type of process constructors, which establish the relationships between the input partitions and the output partitions. The first step of the ForSyDe abstraction is to obtain a ForSyDe model in which the different processes and signals are identified. In order to obtain this abstract model, a direct mapping between *ConcurrencyResource*-processes and *CommunicationMedia*-signals is established. Figure 6 shows the C&C abstract model of Figure 5 using ForSyDe processes and signals. Therefore, with this first abstraction, the ForSyDe C&C system structure is obtained.

There is a particular case related to the ForSyDe abstraction of the *CommunicationMedia*-signal. Assume that in *channel_6* of the example in Figure 4 another MARTE stereotype has been applied, specifically the <<ConcurrencyResource>> stereotype. In this way, the communicating element has the characteristic of performing a specific functionality. This combination of *concurrency resource* and *communication media* semantics can be used in order to model system elements that transmit data and, moreover, perform a transformation of this data. The ForSyDe representation of this kind of channels consists in a process that represents the functionality associated with the channel and a signal that represents the output data generated by the channel after the input data is computed.

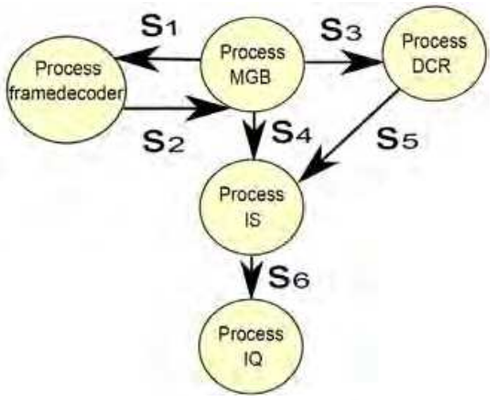



Fig. 6. ForSyDe representation of the C&C model of the Figure 5.

### 4.4 Concurrency resource's behaviour description

A concurrent element can be described by a finite state machine where in each state the concurrent element receives inputs, computes these inputs and calculates their new state and the corresponding outputs. The structure of the behaviour of each concurrency resource is modelled by means of an Activity Diagram. The activity diagram can model the complete resource behaviour. In this case, there is no clear identification of the class states; the states executed by the class during its execution are implicit. Activity diagrams represent activity executions that are composed of single steps to be performed in order to model the complete behaviour of a particular class. These activities can be composed of single actions that represent different behaviours, related to method calls or algorithm descriptions. In this case, the complete behaviour captured in an activity diagram can be structured as a sequence of states fulfilling the following definition: each state is identified as a stage where

the concurrency resource receives the data from its environment; these data are computed by an atomic function, producing the corresponding output data. Therefore, in the most general approach, an implicit state in an activity diagram is determined between two waiting stages, that is, between two stages that represent input data. In this kind of stages, the concurrency resource has to wait until the required data are available in all the inputs associated with the corresponding function. In the same way, if code were directly written, an equivalent activity diagram could be derived. Additionally, the behavioural modelling of the concurrent resources can be modelled by an explicit UML finite state machine. This UML diagram is focused on which states the object covers throughout its execution and the well-defined conditions that trigger the transitions among these states (the states are explicitly identified). Each UML state can have an associated behaviour denoted by the label do. This label identifies the specific behaviour that is performed as long as the concurrent element is in the particular state. Therefore, in order to describe the functionality in each state, UML activity diagrams is used.

Figure 7 shows the activity diagram that captures the functionality performed by the *concurrency resource* of the *IS* component. According to the aforementioned internal state definition, this diagram identifies two states; one state where the *concurrency resource* is only initialized and another state where the tuple data-consumption/computation/data generation is modelled. The data consumption is modelled by a set of *AcceptEventAction.* In the general case, this UML action represents a service call owned by a *communication media* from which the data are required. Then, these data are computed by the atomic function *Scan*. The data generated from this computation (in this case, *data3*) are sent to another system component; the sending of data is modelled by *SendObjectAction* that represents the corresponding service call for the computing data transmissions.

Apart from the UML elements related to the data transmission and the data computation, another set of UML elements are used in order completely specify the functionality to be modelled. The fork node () establishes concurrent flows in order to enable the modelling of data inputs required from different channels in the same state. The UML pins (the white squares) associated to the *AcceptEventAction,* function *Scan* and *SendObjectAction* represent the data received from the communication, the data required/generated by the atomic function execution and the data sending, respectively. An important characteristic needed to define the *concurrency resource* functionality behaviour is the number of data required/generated by a specific atomic function. This characteristic is denoted by the multiplicity value. Multiplicity expresses the minimum and the maximum number of data that can be accepted by or generated from each invocation of a specific atomic function. Additionally, the minimum multiplicity value means that some atomic functions cannot be executed until the receipt of the minimum number of data in all atomic function incoming edges. In Figure 7, the multiplicity values are annotated in blue UML comments.

As was mentioned, *concurrent resource* behaviour is composed of pure functionality represented by atomic functions and *communication media* accesses; the structure of the behaviour of a *concurrency resource* specifies how pure functionality and communication accesses are interlaced. This structure is as relevant as the C&C structure, since both are involved in the executive semantics of the process network.

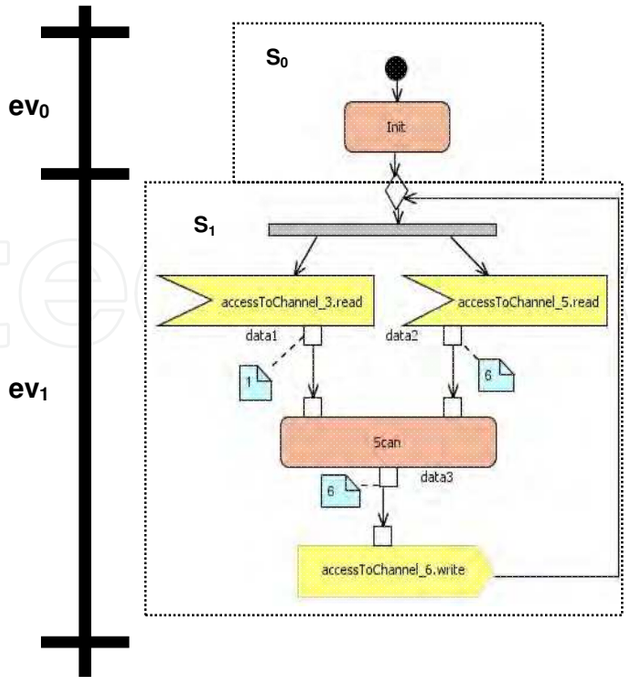



Fig. 7. Activity diagram that describes the functionality implemented by the IS component.

### 4.5 ForSyDe representation of concurrency resource functionality modelling

In the behavioural model in Figure 7 two implicit states ($S_0$ and $S_1$) can be indentified. The activity diagram implicit states are represented as $\omega_j$ in ForSyDe. A state $\omega_j$ is understood to be a state composed of two different states, $P_j$ and $D_j$. In the general case, $P_j$ denotes segments of the behavioural description that are between two consecutive waiting stages. In this case, such waiting stages are identified by two consecutive sets of *AcceptEventActions*. Therefore, $P_j$ corresponds to the basic structure described in the previous section. $D_j$ expresses all internal values that characterize the state. The change in the internal state of a *concurrency resource* is denoted by the *next state function* $g((a_1 \ldots a_n), \omega_j) = \omega_{j+1}$ where $\omega_j$ represents the current state and $a_1 \ldots a_n$ the input data consumed in this state. The function *g()* calculates both $D_{j+1}$ and $P_{j+1}$.

The atomic function implemented in a state $\omega_j$ (for instance, in the example in Figure 7 the function *Scan*) is represented by the ForSyDe *output function* $f_i()$. This function generates the outputs (represented as the subsignals $a'_1 \ldots a'_m$) as a result of computing the data inputs.

The multiplicity values of the input and output data sequences are abstracted by a *partition function* $\nu$ :

$$\text{Input partition functions} \quad \begin{aligned} \nu_1(z) &= \gamma(\omega_i) = p \\ &\ldots \\ \nu_n(z) &= \gamma(\omega_i) = q \end{aligned} \qquad (8)$$

$$\forall z, i \in \mathbb{N}_0 \wedge \{p, q\} \in \mathbb{N}$$

$$\text{Output partition functions} \quad length(f_i(a_1 \ldots a_n), \omega_i) = \begin{cases} \nu'_1(z) = length(a'_1) = a \\ \ldots \\ \nu'_M(z) = length(a'_M) = b \end{cases} \tag{9}$$

$$\forall z, i \in \mathbb{N}_0 \wedge \{a, b\} \in \mathbb{N}$$

A *partition function* enables a signal partition π(v,s), that is, the division of a signal *s* into a sequence of sub-signals $a_i$. The partition function denotes the amount of data consumed/produced in each input/output in each ForSyDe process computation, referred to as evaluation cycle.

The data received by the concurrency resource through the *AcceptEventActions* are represented by the ForSyDe signal $a_1 \ldots a_n$. Regarding the data transmitted through *SendObjectActions*, they are represented by $a'_1 \ldots a'_m$.

In addition, the behavioural description has a ForSyDe time interpretation; Figure 7 corresponds to two evaluation cycles ($ev_0$ and $ev_1$) in ForSyDe. The corresponding time interpretation can be different depending on the specific time domain. These evaluation cycles will have different meanings depending on which MoC the designer desires to capture in the models. In this case, the timing semantics of interest is the untimed semantics.

## 5. UML/MARTE-SystemC mapping

The UML/MARTE-SystemC mapping enables the generation of SystemC executable code from UML/MARTE models.

This mapping enables the association of a corresponding SystemC executable code which reflects the same concurrency and communication structure through processes and channels. Similarly, the SystemC code can reflect the same hierarchical structure as the MARTE model by means of modules, ports, and the different types of SystemC binding schemes (port-port, channel-port, etc). However, other mapping alternatives maintaining the semantic correspondence, using port- export connections, are feasible thanks to the ForSyDe formal link. Figure 8 shows the first approach to the UML/MARTE-SystemC mapping regarding the C&C structure and the system hierarchy. The correspondence among the system hierarchy elements, component-module and port-port, is straightforward. In the same way, the correspondence *concurrency resource*-process is straightforward. A different case is the communicating elements. As a general approach, a communication media corresponds to a SystemC channel. However, the type of SystemC channel depends on the communication semantics captured in the corresponding *communication media*. As can be seen in (Peñil et al., 2009), depending on the characteristics allocated to the *communication media*, different communication semantics can be identified in UML/MARTE models which implies that the SystemC channel to be mapped should implement the same communication semantics.

Regarding the functional description, the *AcceptEventActions* and *SendObjectActions* are mapped to channel accesses. If channel instances are beyond the scope of the module, the accesses to them become port accesses. The multiplicity value of each data transmission in

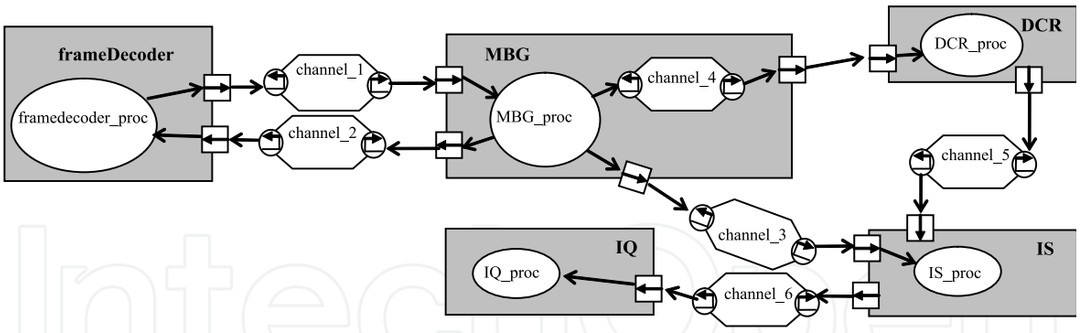



Fig. 8. SystemC representation of the UML/MARTE model in Figure 4.

the activity diagram corresponds to multiple channel accesses (of a single data value) in the SystemC code. Execution of pure functionality captured as atomic functions represents the individual functions that compose the complete *concurrency resource* functionality. The functions can correspond to a representation of functions to be implemented in a later design step according to a description attached to this function or pure C/C++ code allocated to the model. Additionally, loops and conditional structures are considered in order to complement the behaviour specification of the *concurrency resource*. Figure 9 shows the SystemC code structure that corresponds to the functional description of Figure 7. Lines (2-3-4) are the declarations of the variables typed as $T_i$ used for communication and computation. Then, an atomic function for initializing some internal aspects of the *concurrency resource* is executed. Line 5 denotes the statement that defines the infinite loop. Line 6 is the data access to the *communication media channel_3*. In this case, the channel access is done through the port *fromMGB*. In the same way, line 7 is the statement for reading the six data from *channel_5* through the port *fromDCR*. The atomic functions *Scan* is represented as a function call, specifying the function parameters (line 9). Finally, the output data resulting from the *Scan* computation (*data3*) are sent through the port *toIQ* by using the *communication media channel_6*.

```
(1) void IS::IS_proc(){
(2)  T1  data1;
(3)  T2  data2[ ];
(4)  T3  data3[ ];
(5)  Init();
(6)  while (true) {
(7)    data1 = fromMGB.read();
(8)    for(int i=0;i<6;i++) data2[i]= fromDCR.read();
(9)    Scan (dat1, data2, data3);
(10)   for(int i=0;i<6;i++) toIQ.write(data3[i]);
(11) }}
```

Fig. 9. SystemC code corresponding to the model in Figure 7.

### 5.1 UML/MARTE-SystemC mapping: ForSyDe formal foundations

As was described, there are similarities which lead to the conclusion that the link of these MARTE and SystemC methodologies is feasible. However, there are obvious differences in

terms of UML and SystemC primitives. Moreover, there is no exact a one to one correspondence, e.g., in the elements for hierarchical structure. Even when correspondence seems to be straightforward (e.g. *ConcurrencyResource* = SystemC Process), doubts can arise about whether every type of SystemC process can be considered in this relationship. A more subtle, but important consideration in the relationship is that the SystemC code is executable over a Discrete Event (DE) timed simulation kernel, which provides the code with low level execution semantics. SystemC channel implementation internally relies on event synchronizations, shared variables, etc, which map the abstract communication mechanism of the channel onto the DE time axis. In contrast, the execution semantics of the MARTE model relies on the attributes of the *communication media* (Peñil et al, 2009) and on CCSL (Mallet, 2008). A common representation of the abstract semantics of the SystemC channel and of the *communication media* is required. All these reasons make the proposed formal link necessary.

The UML/MARTE-SystemC mapping enables the generation of SystemC executable code from UML/MARTE models. The transformation process should maintain the C&C structure, the behaviour semantics, and the timing information captured in the UML/MARTE models in the corresponding SystemC executable model. This information preservation is supported by ForSyDe, which provides the required semantic consistency. This consistency is provided by a common formal annotation that captures the previous relevant information that characterizes the behaviour of a *concurrency resource* and additional relevant information such as the internal states of the process, the atomic functionality performed in each state, the inputs and the number of inputs required for this atomic functionality to be performed and the resulting data generated outputs from this atomic function execution.

An important characteristic is the timing domain. This article is focused on high-level (untimed) UML/MARTE PIMs. In the untimed models, the time modelling is abstracted as a causality relation; the events communicated by the concurrent elements do not contain any timing information. An order relation is denoted; the event sent first by a producer is received first by a consumer, but there is no relation among events that form different signals. Additionally, the computation and the communication take an arbitrary and unknown amount of time.

Figure 10 shows the ForSyDe abstract, formal annotation of the *IS concurrency resource* behaviour description and the functional specification of the SystemC process *IS_proc*. Line 1 specifies the type of *processor constructor*; in this case the *processor constructor* is a *mealyU*. The U suffix denotes untimed execution semantics. The *mealyU* process constructor defines a process with internal states that take the output function *f()*, the next state functions *g()*, the function $\gamma()$ for defining the signal partitions, and the initial state $\omega_0$ as arguments. In general $\gamma()$, *f()* and *g()*are state-dependent functions. In this case, the abstraction splits *f()*, *g()* and $\gamma()$ into state-independent functions. The function $\gamma()$ is the function used to calculate the new partition functions $v_{sk}$ of the inputs signals. Specifically, output function *f()* of the *IS* process is divided into 2 functions corresponding to the two internal state that the concurrency resource has. The first output function $f_0()$ models the *Init()* function; the output function $f_1()$ models the function *Scan()*. In this function, the partition functions $v_{sk}$ of each input data required for the computing of the *Scan()* (line [7]) are annotated. Line [9] represents the partition function of the resulting output signal $s'_1$. In the same way as in the case of the

function *f()*, next state of the function *g()* is divided into 2 functions, in order to specify the state transitions (lines [5] and [10]) identified in the activity diagram. The data communicated by the *IS concurrent resource data1*, *data2*, *data3* are represented by the signals $S_1$ and $S_2$ for the inputs (data1, data2) and $S'_1$ for the output signal *data3*. The implicit states identified in the activity diagram $St_0$ and $St_1$ are abstracted as the states $\omega_0$ and $\omega_1$, respectively.

[1] IS = mealyU($\gamma$, g, f, $\omega_0$)
[2] IS ($s_1$, $s_2$) = <$s'_1$>

[3] if ($state_i = \omega_0$) then
[4] $f_0()_i$ = Init()
[5] $state_{i+1} = g(\omega_0) = \omega_1$
[6] elseif ($state_i = \omega_1$)

[7] $\begin{cases} \nu_{s1}(i) = 6 \text{ , } \pi(\nu_{s1}, s_1) = <a1_i> \\ \nu_{s2}(i) = 1 \text{ , } \pi(\nu_{s1}, s_1) = <a2_i> \end{cases}$

[8] $a1'_i = f_1(a1_i, a2_i) = Scan(a1_i, a2_i)$
[9] $\nu_{s'1}(i) = 6.\ \pi(\nu_{s'1}, s'_1) = < a1'_i>$
[10] $state_{i+1} = g(\omega_1) = \omega_1$

Fig. 10. ForSyDe annotation of the UML/MARTE model in Figure 7 and the SystemC code in Figure 9.

According to the definition of evaluation cycle presented in section 3, both implicit states that can be identified in the activity diagram shown in Figure 7 correspond to a specific ForSyDe evaluation cycle (*ev0* and *ev1*).

Therefore, the abstract, formal notation shown in Figure 10 captures the same, common behaviour semantics modelled in Figure 7 and specified in Figure 9, and, thus, provides consistency in the mapping between UML/MARTE and SystemC in order to enable the later code generation (Figure 11).

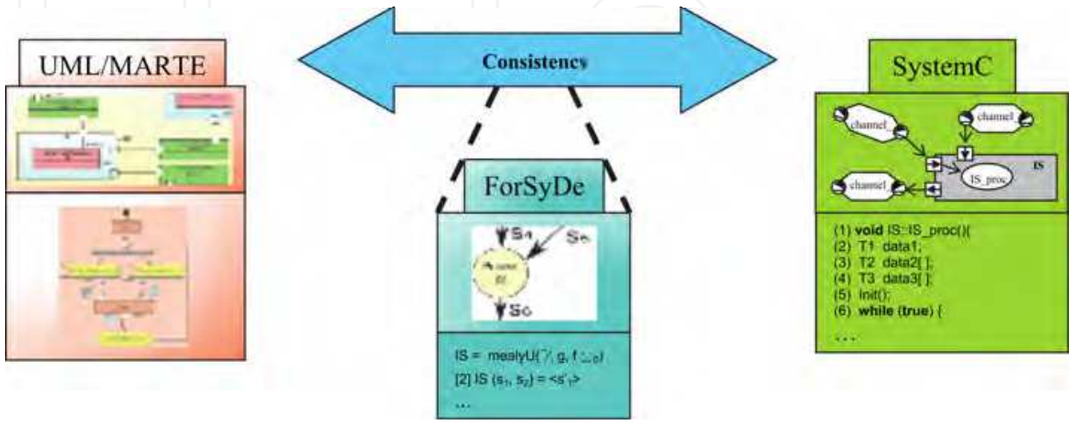



Fig. 11. Representation of mapping between UML/MARTE and SystemC formally supported by ForSyDe.

### 5.2 Formal support for untimed UML/MARTE-SystemC models

The main problem when trying to define a formal mapping between MARTE and SystemC is to define the untimed semantics of a DE simulation language such as SystemC. Under this untimed semantics, the strict ordering of events imposed by the DE simulation mechanism of SystemC's simulation kernel has to be relaxed. In principle, the consecutive events in a particular SystemC object (a channel, accesses to a shared variable, etc.) should be considered as totally ordered as they originate from the execution of a sequential algorithm. Any change in this order in any implementation of the algorithm should be based on a sound optimization methodology or should be clearly explained by the designer. Events in objects corresponding to different concurrent processes related by causal dependencies are also ordered and, again, any change should be fully justified. However, events in objects corresponding to different concurrent processes without any causal dependency can be implemented in any order. This is the flexibility required by the design process in order to ensure optimal implementations under the imposed design constraints.

As was commented previously, SystemC processes and MARTE *concurrency resources* can be directly abstracted as ForSyDe processes. Nevertheless, and in the most general case, the abstraction of a SystemC communication mechanism and the *communication media* relating two processes is more complex. The type of communication in this article is addressed through channels and shared variables. When the communication mechanism fulfils the required conditions, then, it can be straightforwardly abstracted as a ForSyDe signal.

The *MGB* component shown in figure 4 is connected to its particular environment through four *communication media*. Assuming that in these *communication media* four different communication semantics can be identified. The *communication media channel_1* represents an infinite FIFO that implements the semantics associated to the KPN MoC. The *channel_3* establishes a rendezvous communication with data transmission. The way to identify the properties that characterize these communication mechanisms in UML/MARTE models was presented in (Peñil et al, 2009). The *channel_2* represents a shared variable and the *channel_4* is a border channel between the domains KPN-CSP. Therefore, the MGB *concurrency resource* is a border process. A border process is a sort of process which channel accesses are connections to different *communication media* that captured different communication semantics. In this way, the AVD system is a heterogeneous entity where different behaviour semantics can exist.

The data transmission dealt with the MGB *concurrency resource* is carried out by means of a different sort of *communication media*: unlimited FIFO, shared memory, rendezvous and a KPN-CSP border channel. Those communication media accesses are denoted by the corresponding *AcceptEventActions* and *SendObjectActions* identified by the port or channel used by the data transmission and the service called for that data transmission (see Figure 1a)). All these communication semantics captured in the UML/MARTE *communication media* have to be mapped to specific SystemC communication mechanism ensuring the semantic preservation. The *communication media channel_1*, *channel_2* and *channel_4* can be mapped to SystemC channels provided by the HetSC methodology (HetSC, 2007). HetSC is a system methodology based on the ForSyDe foundations for the creation of formal execution specifications for heterogeneous systems. Additionally, HetSC provides a set of communications mechanisms required to implement the semantics of several MoCs. Therefore, the mapping process from the previous *communication media* to the SystemC

channels ensures the semantic equivalence since HetSC provides the required SystemC channels that implement the same communication semantics captured in the corresponding *communication media*. Additionally, these *communication media* fulfil, by construction, the condition that the data obtained by the consumer process are the same and in the same order as the data generated by the producer process. In this way, they can be abstracted as a ForSyDe signal which implies that the *communication media*-SystemC channel mapping is correct-by-construction. As an example of SystemC channel accesses, in Figure 12 b), line (5) denotes a channel access through a port and line (7) specifies a direct channel access.

An additional application of the extracted ForSyDe model is the generation of some properties that the SystemC specification should satisfy under any dynamic condition in any feasible testbench. Note that the ForSyDe model is static in nature and does not include the synchronization and firing mechanism used by the SystemC model. In the example of *MGB* component, a mechanism for communication among processes can be implemented through a shared variable, specifically the *channel_2*. Nevertheless, the communication of concurrent processes through shared variables is a well-known problem in system engineering. As the SystemC simulation semantics is non-preemptive, protecting the access to the shared variables does not make any difference. However, this is an implementation issue when mapping SystemC processes to SW or HW. A variable shared between two SystemC processes correctly implements a ForSyDe signal when the following conditions apply:

1. Every data token written by the producer process is read by the consumer process.
2. Every data token written by the producer process is read only once by the consumer process.

In some cases, in order to simplify the design, the designer may decide to use the shared variable as local memory. As commented above, this problem can be avoided by renaming. A new condition can be applied:

1. If a consumer uses a shared variable as local memory, no new data can be written by the producer until after the last access to local memory by the consumer, that is, during the local memory lifetime of the shared variable.

Additionally, other conditions have to be considered in order to enable a ForSyDe abstraction to be obtained which provides properties to be satisfied in the system design. Another condition to be considered in the *concurrent resource* behaviour description is the use of fork nodes and thus, the modelling of the internal concurrency in a concurrent element. As a design condition, the specification of internal concurrency is not permitted in the *concurrency resource* behaviour (except for the previously mentioned modelling of the data requirements from different inputs). The behaviour description consists of a sequence of internal states to create a complete activity diagram that models the *concurrent resource* behaviour. As a general first approach, it is possible to use the fork node to describe internal concurrent behaviour of a concurrent element if and only if the corresponding inputs and outputs of each concurrent flow are univocal. Among several concurrent flows, it is essential to know from which inputs the data are being taken and to which the outputs are being sent; in a particular state, only one concurrent flow can access specific communication media.

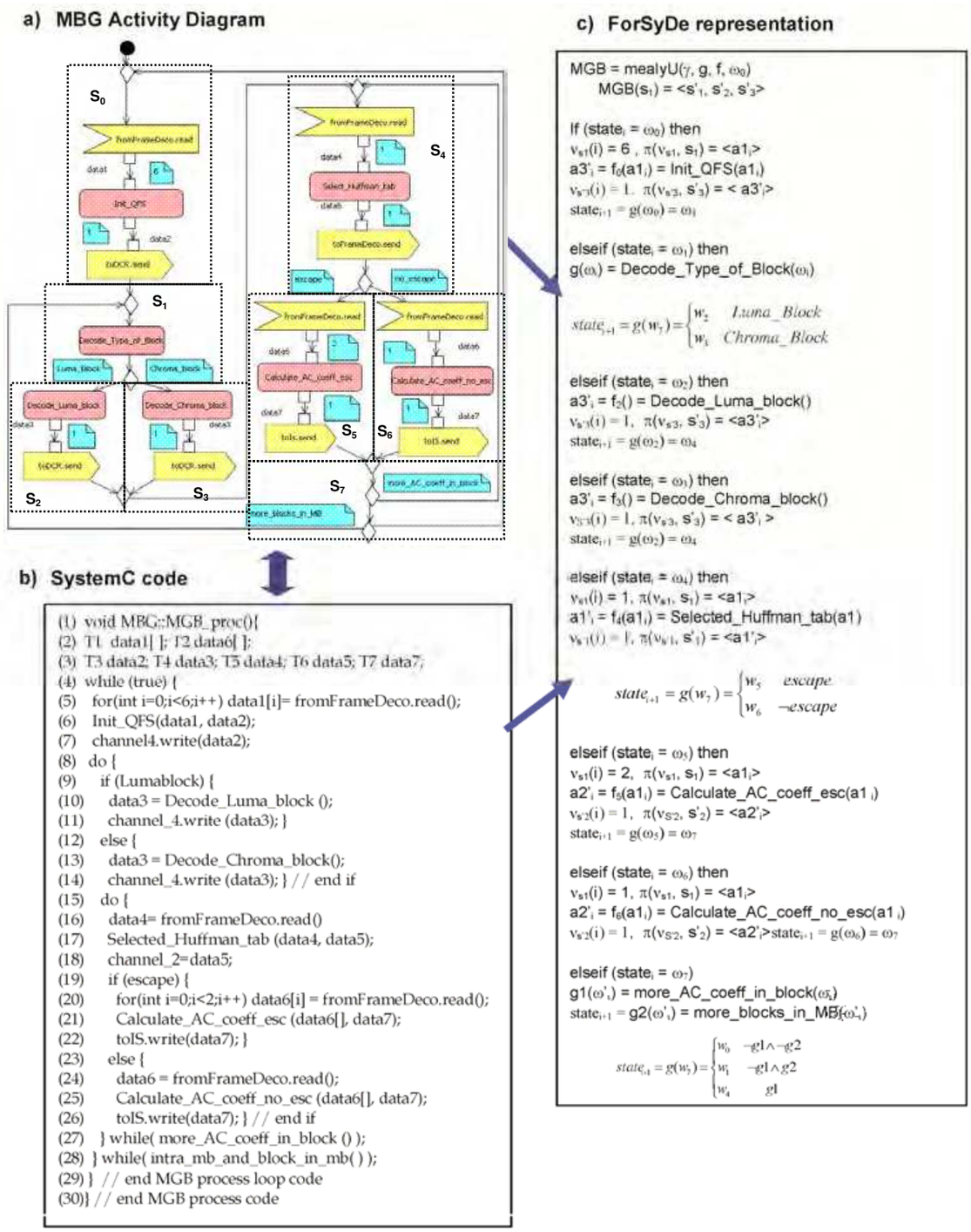



Fig. 12. ForSyDe abstraction (c) of the MBG *concurrency resource* functionality model (a) and its corresponding SystemC code (b).

Another modelling condition that can be considered in the *concurrency resource* behaviour description is the specification of the multiplicity values of the data inputs and outputs. This multiplicity specification has to be explicit and unequivocal, that is, expressions such as [1…3] are not allowed. A previous multiplicity specification is not consistent with the ForSyDe formalization since ForSyDe defines that in each process state, each input and output partition is well defined. The multiplicity specification [a…b] presents indeterminacy in order to define the process behaviour; it is not possible to know univocally the number of data required/produced by a computation. This fact can yield an inconsistent functionality and, thus, can present risks of incorrect performance.

As was mentioned before, not only the communication semantics defined in the communication media is necessary to specify the behaviour semantics of the system, but the way that each communication access is interlaced with pure functionality is also required in order to specify the execution semantics of the processes network. The *communication media channel_3* implements a rendezvous communication among the *MGB concurrency resource* and the *IS concurrency resource* which involves a synchronization and, thus, a partial order in the execution of functions of the two processes. The atomic function *Scan* shown in Figure 7 requires a datum provided by the communication media *channel_3*. This data is provided when either the function *Calculate_AC_coeff_esc* has finished or when the function *Calculate_AC_coeff_no_esc* has finished, depending on which internal state the *MGB concurrency resource* is in. In the same way, the *MGB concurrency resource* needs the *IS concurrency resource* to finish the atomic function *Scan()* in order to go on with the block computation. In this way, the two processes synchronize their independent execution flows, waiting for each other at this point for data exchange. Therefore, besides the semantics captured in the *communication media*, the way the calls to this *communication media* and the computation stages are established in order to model the *concurrency resource*'s behaviour defines its execution semantics, affecting the behaviour of others *concurrency resources*.

The ForSyDe model is a formal representation that enables the capture of the relevant properties that characterize the behaviour of a system. Figure 12 c) shows the ForSyDe formal annotation of the functional model of the MGB *concurrency resource*'s behaviour shown in Figure 12 a) and the SystemC code in Figure 12 b), which is the execution specification of the previous UML/MARTE model. This ForSyDe model specifies the different internal states that can be identified in the activity diagram in Figure 12 a) (all of them identified by a rectangle and the annotation $S_i$). Additionally, ForSyDe formally describes all data requirements for the computations, the functions executed in each state, the data generated in each of these computations and the conditions for the state transitions. This relevant information defines the *concurrency resource*'s behaviour. Therefore, the ForSyDe model provides an abstract untimed semantics associated with the UML/MARTE model which could be used as a reference model for any specification generated from it, specifically, a SystemC specification, in order to guarantee the equivalence between the two system representations.

## 6. Conclusions

This chapter proposes ForSyDe as a formal link between MARTE and SystemC. This link is necessary to maintain the coherence between MARTE models and their corresponding

SystemC executable specifications, in order to provide safe and productive methodologies integrating MDA and ESL design methodologies. Moreover, the chapter provides the formal foundations for enabling this ForSyDe-based link between PIM UML/MARTE models and their corresponding SystemC executable code. The most immediate application of the results of this work will be in the automation of the generation of heterogeneous executable SystemC specifications from untimed UML/MARTE models which specify the system concurrency and communication structure and the behaviour of concurrency resources.

## 7. Acknowledgments

This work was financed by the ICT SATURN (FP7-216807) and COMPLEX (FP7-247999) European projects and by the Spanish MICyT project TEC 2008-04107.

## 8. References

[1] Andersson, P. & M.Höst. (2008). "UML and SystemC a Comparison and Mapping Rules for Automatic Code

[2] Generation", in E. Villar (ed.): "Embedded Systems Specification and Design Languages", Springer, 2008.

[3] Bocchio, S.; Riccobene, E.; Rosti, A. & Scandurra, P. (2008). "An Enhanced SystemC UML Profile for Modeling at

[4] Transaction-Level", in E. Villar (ed.): "Embedded Systems Specification and Design Languages", Springer, 2008.

[5] Ecker, W.; Esen, V. &, Hull, M. (2006). Execution Semantics and Formalisms for Multi-Abstraction TLM Assertions. In Proc. of MEMOCODES'06. Napa, California. July, 2006.

[6] Eshuis, R. & Wieringa, R. (2001). "A Formal Semantics for UML Activity Diagrams–Formalizing Workflow Models",

[7] CTIT Technical Reports Series (01-04).

[8] Falk, J.; Haubelt, C. & Teich, J. (2006). "Efficient Representation and Simulation of Model-Based Designs in SystemC", in proc. of FDL'2006, ECSI, 2006.

[9] Herrera, F & Villar, E. (2006). "A framework for Embedded System Specification under Different Models of Computation in SystemC", in proc. of the Design Automation Conference, DAC'2006, ACM, 2006.

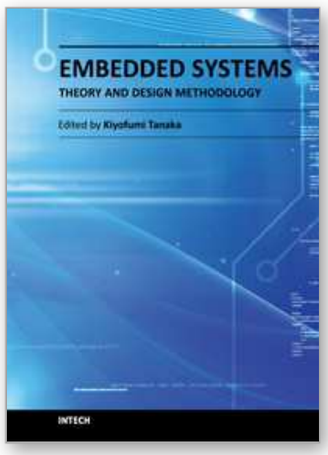
[10] Hoare, C. A. R. (1978). Communicating sequential processes. Commun. ACM 21, 8. 1978.

[11] Jang, E. S.; Ohm, J. & Mattavelli, M. (January 2008). Whitepaper on Reconfigurable Video Coding (RVC). ISO/IEC JTC1/SC29/WG11 N9586. Antalya, Turkey. Available in http://www.chiariglione.org/mpeg/technologies/mpb-rvc/index.htm.

[12] Jantsch, A. (2004). Modeling Embedded Systems and SoCs. Morgan Kaufmann Elsevier Science. ISBN 1558609253.

[13] Kahn, G. (1974). The semantics of a simple language for parallel programming. In Proceedings of the International Federation for Information Processing Working Conference on Data Semantics.
[14] Kreku, J. ; Hoppari, M. & Kestilä, T. (2007). "SystemC workload model generation from UML for performance simulation", in proc. of FDL'2007, ECSI, 2007.
[15] Kroening, D. & Sharygna, N. (2005). "Formal Verification of SystemC by Automatic Hardware/Software Partitioning", in
[16] proc. of MEMOCODES'05.
[17] Lavagno, L.; Martin, G. & Selic, B. (2003). UML for real: design of embedded real-time systems. ISBN 1-4020-7501-4.
[18] Mallet, F. (2008). "Clock constraint specification language: specifying clock constraints with UML/MARTE", Innovations in Systems and Software Engineering, V.4, N.3, October, 2008.
[19] Maraninchi, F.; Moy, M. & L. Maillet-Contoz. (2005). "Lussy: An Open Tool for the Analysis of Systems-on-a-Chip at the Transaction Level", Design Automation of Embedded Systems, V.10, N.2-3, 2005.
[20] Moy, M.; Maraninchin, F. & Maillet-Contoz, L. (2008). "SystemC/TLM Semantics for Heterogeneous System-on-Chip Validation", in proc. of NEWCAS and TAISA Conference, IEEE, 2008.
[21] Mueller, W.; Ruf, J.; Hoffmann, D.; Gerlach, J.; Kropf, T. & W. Rosenstiel. (2001). "The Simulation Semantics of SystemC", in proc. of Design, Automation and Test in Europe, DATE'2001, IEEE, 2001.
[22] Peñil, P; Medina, J. & Posadas, H. & Villar, E. (2009). "Generating Heterogeneous Executable Specifications in SystemC from UML/MARTE Models", in proc. of the 11th Int. Conference on Formal Engineering Methods, IEEE, 2009.
[23] Piel, E.; Attitalah, R. B.; Marquet, P.; Meftali, S. ; Niar, S.; Etien, A.; Dekeyser, J.L. & P. Boulet. (2008). "Gaspard2: from MARTE to SystemC Simulation", in proc. of Design, Automation and Test in Europe, DATE'2008, IEEE, 2008.
[24] UML Specification v2.3. (2010).
[25] UML Profile for MARTE, v1.0. (2009).
[26] MDA guide, Version 1.1, June 2003.
[27] Open SystemC Initiative. www.systemc.org.
[28] Raudvere, T.; Sander, I. & Jantsch, A. (2008). "Application and Verification of Local Non Semantic-Preserving Transformations in System Design", IEEE Trans. on CAD of ICs and Systems, V.27, N.6, 2008.
[29] Salem, A. (2003). "Formal Semantics of Synchronous SystemC", in proc. of Design, Automation and Test in Europe, DATE'2003, IEEE, 2003.
[30] Störrle, H. & Hausmann, J.H. (2005). "Towards a Formal Semantics of UML 2.0 Activities", Software Engineering Vol. 64.
[31] Taha, S.; Radermacher, A.; Gerard, S. & Dekeyser, J. L. (2007). "MARTE: UML-based Hardware Design from Modeling to Simulation", in proc. of FDL'2007, ECSI 2007.
[32] Traulsem, C.; Cornet, J.; Moy, M. & Maraninchi, F. (2007). "A SystemC/TLM semantics in PROMELA and its possible Applications", in proc. of the Workshop on Model Checking Software, SPIN'2007, 2007.

[33] Vidal, J.; de Lamotte, F.; Gogniat, G.; Soulard, P. & Diguet, J.P. (2009). "A Code-Design Approach for Embedded System Modeling and Code Generation with UML and MARTE", proc. of the Design, Automation & Test in Europe Conference, DATE'09, IEEE 2009.




**Embedded Systems - Theory and Design Methodology**
Edited by Dr. Kiyofumi Tanaka

ISBN 978-953-51-0167-3
Hard cover, 430 pages
**Publisher** InTech
**Published online** 02, March, 2012
**Published in print edition** March, 2012

Nowadays, embedded systems - the computer systems that are embedded in various kinds of devices and play an important role of specific control functions, have permitted various aspects of industry. Therefore, we can hardly discuss our life and society from now onwards without referring to embedded systems. For wide-ranging embedded systems to continue their growth, a number of high-quality fundamental and applied researches are indispensable. This book contains 19 excellent chapters and addresses a wide spectrum of research topics on embedded systems, including basic researches, theoretical studies, and practical work. Embedded systems can be made only after fusing miscellaneous technologies together. Various technologies condensed in this book will be helpful to researchers and engineers around the world.



**InTech Europe**
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

**InTech China**
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen