A general approach to the interoperability of HetSC and SystemC-AMS

F. Herrera⋆, E. Villar⋆, C. Grimm†,
M. Damm† and J. Haase †
⋆University of Cantabria, Spain
†Technical University of Vienna, Austria

Abstract

This paper studies and proposes a joint use of SystemC-AMS and HetSC (**Het**erogeneous **S**ystem**C**) heterogeneous specification methodologies. This enables an efficient support of a wide range of Models of Computation (MoCs). In this way, SystemC can be used for the complete specification of embedded systems, which are increasingly heterogeneous, since they include the software control part, digital hardware accelerators, the analog front-end, etc. This paper identifies and solves the syntactical and semantical issues involved in the cooperation of the SystemC-AMS and HetSC specification methodologies. This includes considering the availability and suitability of the MoC interface facilities provided by both methodologies, especially those of SystemC-AMS, which will be proposed for future standardization. Some practical aspects, such as the compatibility and installation of their respective libraries and the definition of the set of MoCs covered are also dealt with.

1 Introduction

[1] Support for heterogeneity has become an important feature for specification methodologies that aim to cope with the current complexity of embedded systems. In this context, heterogeneity is the ability of the specification methodology to enable the building of models with parts specified under different MoCs[LSV98].

Each design domain adopts a specification methodology which usually corresponds to a specific model of computation (MoC). One of the most characteristic points associated with the MoC is the handling of time. For instance, analog models (Continuous Time (CT) models[Jan03]) handle strict-time information, that is, specification events have an associated time tag representing physical time and fixing strict order relationships among them. In contrast, concurrent software models often neglect such detail in the time domain and consider only partial order relationships among the events associated to the code.

[1]Work supported by the FP6-2005-IST-5 european project.

The development of a system-level heterogeneous specification methodology is, to a great extent, a unification work. Some works developed interfaces between different languages, i.e., to connect hardware description languages (HDLs) with high-level programming languages [Gup02]. Later on, the focus was on the specification framework as the common point. Relevant examples are Metropolis [DDM+07] and Ptolemy II [BLL+05]. These frameworks enable specification under different MoCs, approaching the separation of computation and communication in different ways. Both provide support for graphical specification, while Java adopts the role of underlying implementation language. Up to now, the focus of this unifying work has tended to be the language itself. The lack of a unified system specification language has been identified as one of the main obstacles bedeviling SoC designers [Gep00]. A common specification language is a major aid in generating a design specification methodology which aims to combine and achieve coherence among traditionally different and separated design approaches.

SystemC has started to play a role as unifying system-level language for embedded system design. Becoming an IEEE standard is a symptom of its acceptance and of a stated syntax and unambiguous semantic for the language constructs which are used by SystemC-based methodologies. In this context, several proposals have appeared for building a heterogeneous specification methodology over SystemC.

This paper shows how the HetSC and SystemC-AMS heterogeneous specification methodologies enable the use of SystemC to build models which comprise a wide spectrum of MoCs. This involves studying and understanding how the existing facilities for MoC connection in the two methodologies can be used and combined to provide improved connections. Firstly, in section 2, the work done in this area regarding SystemC is reviewed, mainly focusing on the HetSC and SystemC-AMS specification methodologies. In section 3, the general issues of their interoperability are explained. First, some practical issues concerning the installation and the scope of the libraries are discussed. Then, how the SystemC-AMS and HetSC constructs are mixed in the same specification is explained. Section 4 provides an illustrative example of the previous concepts. Section 5 ends with the main conclusions of this work.

2 Related Work

Although the SystemC core language supports hardware specification (RTL and Behavioral) and a generic Discrete Event (DE) modelling, there is a set of MoCs which are not sufficiently supported by the core language. Such support must include new specification facilities, MoC rule checkers, report tools, etc. Several works have attempted to cover such deficiencies. In the following paragraphs, these works are overviewed.

SystemC-AMS [VGE04] is a specification methodology developped by the OSCI SystemC-AMS working group which provides support for analog and mixed-signal specification. This involves supporting the Synchronous Dataflow (SDF), discrete-time (DT) and continuous time (CT) MoCs. Among the CT MoCs, it is possible to specify linear behavioral models by means of transfer functions (TF). Currently, two views are supported for TFs: the numerator-denominator (ND) view and the zero-pole (ZP) view. In addition, the specification of linear electrical networks (LEN), which enable a circuit level description is also supported.

SystemC-AMS is extensible by other models of computation through a synchronization layer. Solvers for the MoCs supported are layered over the synchronization layer. The design of the synchronization layer of SystemC-AMS and the MoCs provided are oriented to a system-level modelling where simulation speed is a more important factor than a very fine simulation accuracy. The synchronization layer supports directed communication and only a simple synchronization; on user specified events or in fixed time steps. In this way, the simulation of linear networks with SystemC-AMS can be orders of magnitudes faster than the more general numerical integration for non-linear networks [HOS$^+$07]. From the specification point of view, SystemC-AMS offers a new set of facilities, such as new kinds of modules (*SCA_SDF_MODULE*), ports(*sca_sdf_in*, *sca_sdf_out*, etc), channels (*sca_sdf_signal*), and other MoC specific facilities, such as the *sca_elec_node*, *sca_elec_port*, etc. Linear behavioral models are embedded in SDF modules, while LENs are enclosed in SystemC modules. SystemC-AMS provides converter ports and facilites to enable different MoCs (i.e. DE with SDF, SDF with LEN, etc) to communicate.

HetSC [HV06] is a methodology for enabling heterogeneous specifications of complex embedded systems in SystemC. MoCs supported include untimed MoCs, such as Kahn Process Networks (KPN), its bounded fifo version (PN), Communicating Sequential Processes (CSP) and Synchronous Dataflow (SDF). Synchronous MoCs, such as Synchronous Reactive (SR) and Clocked Synchronous (CS) and the timed MoCs already supported in SystemC are also included. HetSC aims at a complete system-level HW/SW codesign flow. Indeed, the methodology has been checked in terms of system-level profiling and software generation [PHF$^+$04].

The HetSC methodology defines a set of specification rules and coding guidelines for each specific MoC, which makes the designer task more systematic. The support of some specific MoCs requires new specification facilities providing the specific semantic content and abstraction level required by the corresponding MoCs. The HetSC library, associated with the HetSC methodology, provides this set of facilities to cover the deficiencies of the SystemC core language for heterogeneous specification. In addition, some facilities of the HetSC library help to detect and locate MoC rule violations and assist the debugging of concurrent specifications. One of the main contributions of HetSC is its efficient support of abstract MoCs (untimed and synchronous). This is because they are directly supported over the underlying discrete event (DE) strict-time simulation kernel of SystemC. New abstract MoCs do not require additional solvers since the new MoC semantic is embedded in the implementation of the new specification facilities (usually channels) related to the abstract MoC. When the new MoC can be abstracted from the DE strict-time MoC, then, it is possible to find a mapping of internal events of the new specification facility, i.e, a channel, over the strict-time axis of the DE base MoC. This makes it feasible to write the implementation of such a channel by using SystemC primitives, such as SystemC events, which control when things happen within the channel and, therefore, in the processes related by the channel.

SystemC-H [PS04] is a methodology that proposes a general extension of the SystemC kernel for the support of different MoCs. The extension would include a solver for each MoC. The current scope of the SystemC-H library covers the SDF and CSP untimed MoCs. For instance, SystemC-H provides a solver for static scheduling of SDF graphs which enables schedulability analysis and provides a 75% speed-up respect to DE [PS04]. However, this extension is not always worthwhile. Indeed, the speed-up for some abstract MoCs can be negligible [PMS04]. For instance, the speed-up decreases to 13% for a mixed DE-SDF example [PS04]. In addition, similar speed-ups were reported for the dynamic approach to SDF for large-grain SDF specifications [HV06].

SysteMoC [FHT06] focuses on providing a methodology with the ability to extract and analyze the MoC employed in the SystemC design. This is understood to be a prerequisite for the rest of the design activities. In order to achieve this, the SysteMoC library provides support for a basic MoC called *Funstate*. Specifications written under this MoC express their communication behavior under the finite state machine (FSM) MoC. This enables the automatic extraction and analysis of the MoC employed, only by analyzing communication FSMs together with the topology of the specification.

3 HetSC/SystemC-AMS Interoperability

3.1 Installation and scope

Figure 1 describes the installation requirements of the SystemC user. Apart from the SystemC core library, the SystemC-AMS and HetSC libraries have to be installed on top of the SystemC core library. There is flexibility with respect to the development platform (i.e, Linux, Unix and Windows-Cygwin are supported).

There is no compatibility problem in the installation of HetSC and SystemC-AMS libraries. In this work, the HetSC library is extended with some specific fa-

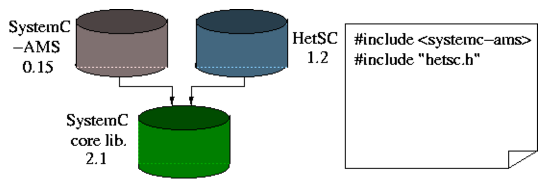

Figure 1: Libraries installed.

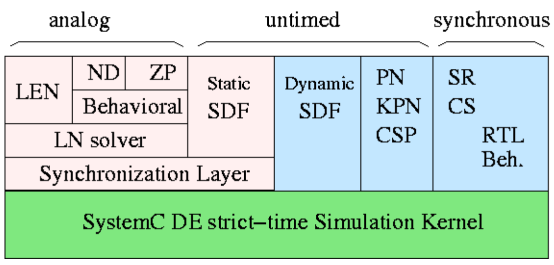

Figure 2: MoCs with SystemC-AMS-HetSC.

cilities for enabling an easier connection of HetSC and SystemC-AMS parts. These HetSC facilities use some SystemC-AMS facilities through forward declarations. This prevents obliging an installation order between HetSC and SystemC-AMS libraries, making the installation procedure easier. Once such an installation has been done, the development system is ready for compiling and executing SystemC specifications written under a wide range of MoCs. The user only has to include the SystemC-AMS and HetSC libraries in the source code of the heterogeneous specification.

Figure 2 shows the supported MoCs. The cooperation of SystemC-AMS and HetSC provides a complementary MoC support. While SystemC-AMS provides support for analog MoCs and static synchronous data flow, untimed and synchronous MoCs are supported by HetSC and SystemC core facilities.

This is also an efficient configuration for the support of a wide spectrum of MoCs. The reason is that specific solvers are provided only for a set of MoCs where the simulation speed up is significant. This set corresponds to analog MoCs where the simulation speed ups can be of orders of magnitude. Bearing in mind the limited speed-ups reported in [PS04, PMS04, HV06], untimed and synchronous MoCs can be satisfactorily supported directly over the SystemC kernel. The exception would be fine grain SDF specifications, where the speed up of a static SDF compared to a dynamic SDF could be significant. Specifications without CT parts but with synchronous hardware (RTL or behavioral) could also justify a cycle-accurate simulator. However, the study of these exceptions is not in the scope of this work.

3.2 Syntactical and Semantical issues

There are some basic issues to consider in a general discussion of the connection between HetSC and SystemC-AMS. From the specification structure point of view, in the specification, two parts will be distinguished. One corresponds to the AMS part, while the other corresponds to the HetSC part. From the syntactical point of view the SystemC part of the specification will be identified by *SCA_SDF_MODULE*s and/or SCA hierarchical modules. The AMS part presents a hierarchical heterogeneity where the underlying MoC is the static synchronous dataflow (SDF) MoC. The other part is the HetSC part, characterized, in general, by an amorphous heterogeneity. This means that the HetSC specification permits mixing MoC facilities in a flat hierarchy. Nevertheless, the HetSC specifier will often make use of module hierarchy for separating parts of the system under different MoCs. Thus, in many cases, module partition will correspond with MoC boundaries.

From the semantical point of view, there is a basic consideration. While HetSC directly relies on the DE strict-time simulation kernel, SystemC-AMS relies on a syncronization layer, which provides support for the solvers. In SystemC-AMS, CT descriptions are always embedded in dataflow clusters [VGE04]. That is, the most important solver is the SDF one which, from the point of view of time semantics, is the basis for the analog MoCs. The time axis in SystemC-AMS is actually sliced by each SDF cluster in strict-time delays called cluster periods, which depend on the cluster period (T) and the rates of the cluster SDF graph. Thus, with respect to the premises of [LM87], the SDF approach of SystemC-AMS is not an untimed SDF. Internally, modules of the cluster can be viewed as a strict-time timed approach to the SDF MoC (denoted as T-SDF here), which enables a static execution of the AMS processes at each cluster period. More important for the purpose of this work, from an external point of view, the cluster can be conceived as a timed-clocked synchronous (CS) block which triggers at each cluster period. Thus, the cluster period must be taken into account to synchronize the DE part with the SystemC-AMS part.

Since every MoC supported by HetSC is abstracted over the DE strict-time simulation kernel and every SystemC-AMS MoC is clustered in the T-SDF MoC, the problem is reduced to providing a DE/AMS connection, which is basically a DE/T-SDF connection. In SystemC-AMS, this connection is focused on the SystemC signal (*sc_signal*). A set of SystemC-AMS specification facilities (*sca_sc2sdf_in*, *sca_sc2sdf_out*, *sca_sc2v*, *sca_sc2r*, etc) are used for such direct connection through the sampling and update of the SystemC signal. Therefore, an immediate conclusion is that these elements can be employed to combine HetSC and SystemC-AMS. This idea can be conveniently exploited and coherently complemented with one of the basic concepts employed in HetSC for the connection of MoCs: the border process. In Figure 3, a HetSC reactive chain [HV06] is composed of a generator process (GP) which triggers a reactive process (RP). This RP is also a border process (BP), since it writes to a SystemC signal channel, provoking the update of its value in the next delta cycle. From the SR MoC point of view, this connection is compatible with the SR rules,

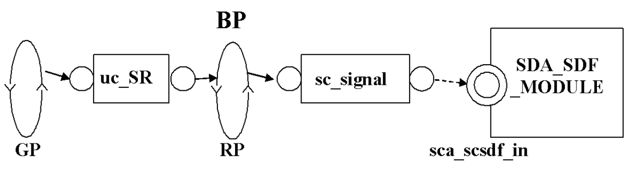

Figure 3: BP connection.

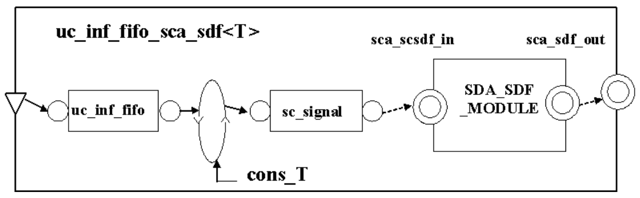

Figure 4: Structure of *uc_inf_fifo_sca_sdf* channel.

specifically with perfect synchrony, since the write access to the SystemC signal is non-blocking.

However, the connection of SystemC-AMS and HetSC can be improved. The set of MoCs abstracted from the DE MoC and supported by HetSC is rich enough to consider specific connections. For instance, the connection of a KPN MoC and a LEN MoC involves fifo channels on one side and electrical nodes on the other side. It would be convenient to count on some connection facility which enables such direct connection, without the intermediation of the SystemC signal (*sc_signal*). The HetSC border channel (BC) can be employed to achieve such direct connection between HetSC and SystemC-AMS. Figure 4 illustrates this. It shows a border channel (*uc_inf_fifo_sca_sdf*), which enables a direct connection between a KPN MoC and a T-SDF MoC. It is built as a hierarchical channel which on the one hand exports the write interface of a *uc_inf_fifo* channel, while on the other hand offers a T-SDF port (*sca_sdf_out*) port. Internally, it uses a border process which consumes fifo tokens, whose values are used to update the internal SystemC signal. The signal is connected to a converter port (*sca_scsdf_in*) of an inner SystemC-AMS module. In addition, BCs provide an scalable way to construct these direct connections since it does not require the SystemC-AMS kernel to be changed. Furthermore, BCs can solve another issue of SystemC-AMS DE/AMS interface primitives. They are based on sampling (read) and updating (write) signals attending the cluster period. However, a KPN part, for instance, only produces or consumes data. Then, some kind of adaptation has to be introduced to convert consumption in sampling (and vice versa) and production in writing (and vice versa). This is not actually defined by the SystemC-AMS connection facilities. Such adaption can be then explicitly written, i.e. in a border process. The BC enable the packaging of such adaptation in a specification primitive. For instance, in the *uc_inf_fifo_sca_sdf* channel, it is defined when to consume fifo tokens by means of a sampling period, which, in general, can be different from the cluster period. The BC can also raise an error if the internal fifo gets empty when a new sampling is given.

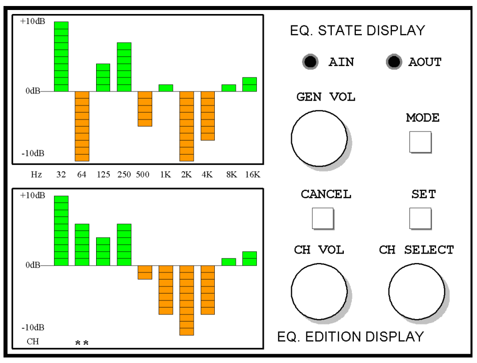

Figure 5: Soundboard System.

4 Example

In order to demonstrate the previous general concepts, an example has been developped. It consists of a soundboard, which is shown in Figure 5. The system has an audio input and an audio output. The audio input undergoes three stage filtering. The first filter is a noise filter to remove any signal component over 22KHz. The second one is a 10-channel equalizer. The last one is an integrator, which, at the same time, controls the general volume and filters the DC component of the audio output. The system has other inputs, as well as the audio input. A dial enables selection of the equalizer channel, while another dial tunes the gain of the selected channel in dBs (in a [-10dB,10dB] range). A state display shows the current state of the equalization, while an edition display shows the currently selected channel and the currently edited equalization profile. This profile is not applied till the *set* button is pressed. Then, the state display changes to reflect this equalization profile. If the *cancel* button is pressed instead, then the edition display and the edition equalization profile return to the initial state (0 dB for every channels). Another dial controls the general gain of the system (also in a [-10dB,10dB] range). It does not depend on the *set* button. That is, its change immediately updates the system gain.

Figure 6 depicts how this has been solved using HetSC and SystemC-AMS together. The system is enclosed in a SystemC module (*soundboard*). This top module contains another SystemC module (*panel_control*), which contains the HetSC part of the system and uses the HetSC library specification facilities. The *soundboard* module also contains three modules which use SystemC-AMS facilities. In this case, the testbench model (*testbench* module) is composed of four modules which only use HetSC facilities.

In Figure 6, the correspondence with the MoCs employed is depicted with dashed lines. In the testbench module, two processes (*left_hand* and *right_hand*) model the handling of dials and buttons of the soundboard. The two processes are synchronized through a rendez-vous channel, to ensure the left hand edits the equalizer profile before the right hand pushes the *set* button and raises the general volume. Because of this, this part is a CSP network. In addition, each of the processes is an autonomous process generating a SR reactive chain. Dial turn and button press are modeled as writes to *uc_SR* channels. The reactive chain which controls the general volume is pure, in that it is composed only of generator and reactive SR processes. The reactive process converts the dial events (turning left or right), which mean plus 1 dB or minus 1 dB, considering the bounds of the [-10dB,10dB] range, in a control SystemC signal which affects the value of a resistor composing the integrator module. A similar thing happens with the channel equalization control. However, here there is not a pure reactive chain, since the two reactive processes are border processes, as they also write to infinite fifo HetSC channels (*uc_inf_fifo*), proper of the KPN MoC. For instance, one is used to pass the new equalization profile to the state display when the *set* button is pressed.

In the analog part, the noise filter is modeled through a SystemC-AMS SDF module (*noise_filter*). This module has an input port, to read the *s_in* external signal which provides the audio samples. It is designed as a second order Butterworth low pass filter with a cut frequency of 22KHz, which is modeled under the LN-TF MoC of SystemC-AMS, using the ND view. The other two blocks are modelled at a circuit level, under the LEN MoC. The *equalizer_array* module encloses an array of 10 equalizer cells. Each of them is an active band pass filter centered at the channel frequency. This filter is described as a circuit with 3 resistors, 2 capacitors and a model of operational amplifier (OA) which considers the gain, the input and output resistance. Each equalizer cell is instantiated taking the capacitor values as the parameter for centering each filter at the channel frequency (32Hz for channel 0, 64Hz for channel 1 and so on till 16KHz for channel 9). The output of each equalizer cell is connected to a resistor instance of type *sca_sc2r*, controlled by one of the signals of the *Rch_ctrl* signal array. These resistors are connected to the same eletrical output node, where the contribution of each equalizer cell is added. This node is used as input to the *integrator* module. This module is also described as a circuit which also instantiates the previously mentioned OA model, a capacitor and a resistor controlled by the *Rgen_ctrl* signal, to control the gain of the integrator and, thus, of the whole system.

In both, the HetSC and SystemC-AMS parts, elements are employed to connect MoCs. For instance, BPs connect KPN and SR MoCs in the HetSC part, and a *sca_sdf2v* instance connects the noise filter to the equalizer array. In Figure 5, the connections between the HetSC and the SystemC-AMS part have been highlighted with thicker arrows. Specifically, the audio input samples are transferred to the *soundboard* module through an instance of the *uc_inf_fifo_sca_sdf*

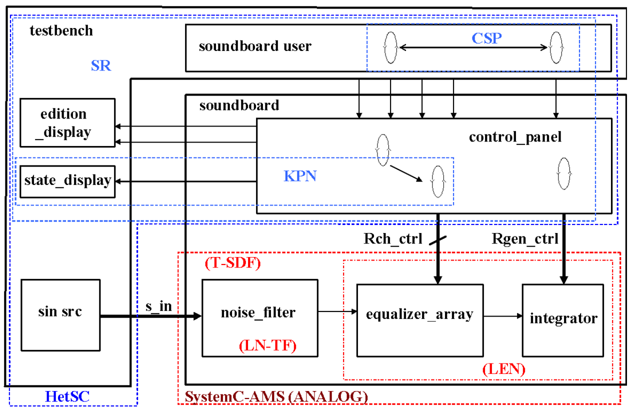

Figure 6: SystemC-AMS-HetSC specification.

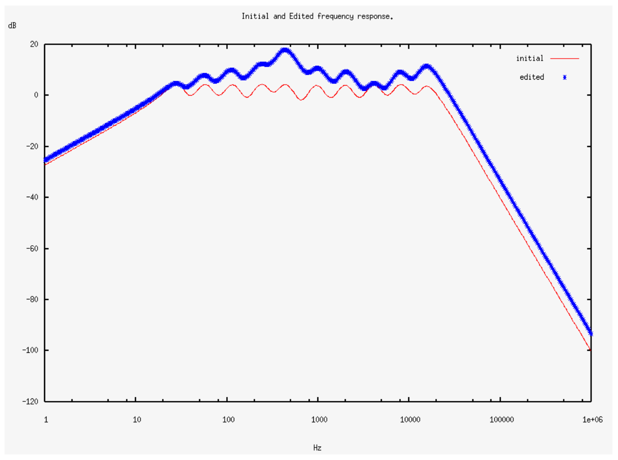

Figure 7: Initial and edited frequency spectrum.

channel introduced in the previous section. This border channel enables a direct connection between the untimed part, which generates the samples, to the SystemC-AMS input converter port of the noise filter. The connection of the SR reactive chains to the LEN part of the model is placed between the lower part of the *control_panel* module and the *equalizer_array* and *integrator* analog modules. For instance, the reactive process triggered by the turn events of the general volume dial is indeed a border process which writes the *Rgen_ctrl* SystemC signal. A similar thing happens with the non pure reactive chain, which drives an array of 10 signals (each one for its corresponding equalizer channel). Each of these signals controls the value of a SystemC-AMS *sca_sc2r* primitive.

A time domain simulation and two frequency analyses have been peformed. The time domain simulation is dumped to data and waveform files. The first frequency analysis is done in the middle of the time domain simulation. At this time, the soundboard response corresponds to that of the initial state (0dB gain for every channel and for the general volume). The second ac analysis is done at the end of the time domain simulation. Then, a manual configuration has already been performed and the *set* button pressed. The result are two data files, whose values are represented in Figure 7. Figure 7 shows the change on the equalization profile. Other outputs of the system are two log files which reflect the activity in the displays.

This specification has been written in just over 2500 SystemC lines including testbench modules. It took around 30 man-hours (ignoring learning time). The simulation time was less than 53 seconds in an Intel P4 2.8GHz, Linux 2.6.3 development platform. This illustrates how fast the system-level specification and analysis of such heterogeneous system can be done using HetSC and SystemC-AMS.

5 Conclusions

This work addresses how the HetSC and SystemC-AMS specification methodologies can be used together. With their cooperation, a wide range of MoCs, from untimed to analog ones, are efficiently covered. This is a key feature in enabling the early system-level specification of embedded systems. The installation and compatibility of the HetSC and SystemC-AMS libraries has been checked. Furthermore, the syntactical and semantical issues related to the connection have been discussed. SystemC-AMS is based on a timed SDF MoC, where AMS clusters can be conceptually seen as timed-clocked synchronous blocks from the DE part. SystemC-AMS provides facilities for this AMS/DE connection which are based on the sampling and update of the SystemC signal. Since HetSC MoCs are abstracted from the underlying DE strict-time MoC, the connection of any HetSC MoC with any System-AMS MoC can be reduced to a SystemC DE/SystemC-AMS connection. Thus, SystemC-AMS facilities for the DE/AMS connection can be used. Moreover, the HetSC border channel can be conveniently used to provide direct connections among specific untimed and synchronous (HetSC) MoCs and analog (SystemC-AMS) MoCs, hiding the intermediation of DE signals in the connection of MoCs that do not employ such specification primitives and encapsulating the detection of error situations which consider the cluster period, the time conditions of HetSC part, etc. Further research on this topic will involve the use of polymorphic signals [SGK05], which will release any manual engagement in the system refinement. Finally, this work implicitly states the need for a formal environment in order to obtain a common understanding of the interoperation of this kind of methodologies.

References

[BLL+05] C. Brooks et al. Ptolemy II: Heterogeneous Concurrent Modeling and Design in Java. Tech. Report, Univ. of California Berkeley, July 2005.

[DDM+07] A. Davare et al. A next-generation design framework for blatform-based design. In *DVCon 2007*, February 2007.

[FHT06] J. Falk, C. Haubelt, and J. Teich. Efficient representation and simulation of model-based designs in SystemC. In *Proc. of FDL'06*, Darmstad, September 2006.

[Gep00] L. Geppert. Electronic Design Automation. *IEEE Spectrum*, 37(1), January 2000.

[Gup02] R. Gupta. HDL/C interface exploration. Tech. Report, ICS Dpt., University of California, California, USA, 2002.

[HOS+07] A. Herrholz et al. ANDRES - ANalysis and Design of Runtime rEconfigurable heterogeneous Systems. In *Proc. of DATE'07*, Nice, April 2007.

[HV06] F. Herrera and E. Villar. A framework for embedded system specification under different models of computation in SystemC. In *Proceedings of DAC'06*, 2006.

[Jan03] A. Jantsch. *Modelling Embedded Systems and SoCs.* Morgan Kaufmann, June 2003.

[LM87] E. A. Lee and D. G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. on Computers*, C-36(1):24-35, 1987.

[LSV98] E. A. Lee and A. Sangiovanni-Vincentelli. A Framework for comparing Models of Computation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(12), December 1998.

[PHF+04] H. Posadas, F. Herrera, V. Fernandez, P. Sanchez, and E. Villar. Single source design environment for embedded systems based on SystemC. *Transactions on Design Automation of Electronic Embedded Systems*, 9(4):293 – 312, December 2004.

[PMS04] H.D. Patel, D. Mathaikutty, and S.K. Shukla. Implementing multi-moc extensions for SystemC: Adding CSP and FSM kernels for heterogeneous modelling. T. Report, FERMAT, Virginia Tech., June 2004.

[PS04] H.D. Patel and S.K. Shukla. *SystemC Kernel Extensions for Heterogeneous System Modeling: A Framework for Multi-MoC Modeling.* Springer, July 2004.

[SGK05] R. Schroll, C. Grimm, and Waldschmidt K. Verfeinerung von mixed-signal systemen mit polymorphen signalen. In *Analog'05.* VDE-Verlag. Berlin, Germany, 2005.

[VGE04] A. Vachoux, C. Grimm, and K. Einwich. Towards analog and mixed-signal SoC design with SystemC-AMS. In *IEEE DELTA'04*, Perth, Australia, 2004.