Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2008, Article ID 738136, 16 pages
doi:10.1155/2008/738136

Research Article

Bridging MoCs in SystemC Specifications of Heterogeneous Systems

Markus Damm,[1] Jan Haase,[1] Christoph Grimm,[1] Fernando Herrera,[2] and Eugenio Villar[2]

[1] *Institute of Computer Technology, Vienna University of Technology, 1040 Vienna, Austria*
[2] *Microelectronics Engineering Group, TEISA Department, University of Cantabria, 39005 Santander, Spain*

Correspondence should be addressed to Jan Haase, haase@ict.tuwien.ac.at

Received 15 October 2007; Revised 21 February 2008; Accepted 14 May 2008

Recommended by Sandeep Shukla

In order to get an efficient specification and simulation of a heterogeneous system, the choice of an appropriate model of computation (MoC) for each system part is essential. The choice depends on the design domain (e.g., analogue or digital), and the suitable abstraction level used to specify and analyse the aspects considered to be important in each system part. In practice, MoC choice is implicitly made by selecting a suitable language and a simulation tool for each system part. This approach requires the connection of different languages and simulation tools when the specification and simulation of the system are considered as a whole. SystemC is able to support a more unified specification methodology and simulation environment for heterogeneous system, since it is extensible by libraries that support additional MoCs. A major requisite of these libraries is to provide means to connect system parts which are specified using different MoCs. However, these connection means usually do not provide enough flexibility to select and tune the right conversion semantic in a mixed-level specification, simulation, and refinement process. In this article, converter channels, a flexible approach for MoC connection within a SystemC environment consisting of three extensions, namely, SystemC-AMS, HetSC, and OSSS+R, are presented.

1. INTRODUCTION

Today's embedded systems mostly consist of digital hardware, analogue hardware and software, a circumstance often referred to as being heterogeneous. The description and simulation of such systems are usually associated to different teams with an expertise in a specific domain (e.g., signal processing, analogue hardware, digital hardware and software). Each team selects the most suitable language and simulation tool for its domain. For instance, digital hardware is described with an HDL and simulated with a cycle-based simulator, while an analogue part is described as a transistor-level schematic and simulated with solver-based tools.

The role of a system engineer in such a distributed work flow is to conceive the specification of the whole system and deliver it to domain specific teams. Often, the specification is a text, that is, written in English, while at other times, it is an abstract model written in a high-level programming language, such as C.

This approach has several drawbacks, where the major one is the late coupling of the different system parts. Specification flaws with respect to the overall system are detected quite late (after the development phase, instead of detecting them after the specification phase). In addition, the use of different languages results in additional difficulties. First, it involves the cumbersome task of coupling simulators, which mostly involves ad hoc solutions for respective simulator pairs. Moreover, it makes the overall system description more difficult to understand. Even if the different languages used have a similar syntax, they sometimes present subtle, but important semantical differences. For instance, the "signal" concept is employed in different metamodels, in RTOS interprocess communications, in the Esterel synchronous language and in VHDL, and is used for the communication of concurrent computations. However, the respective semantics of the "signal" happen to be very different. Thus, it is important to ensure that the system designer is able to deliver an unambiguous specification, where the meaning of each part is clear for all the agents involved in the design.

An alternative is to provide the system engineer with a methodology to produce an executable system-level specification whose constructs are expressive, general, and specific

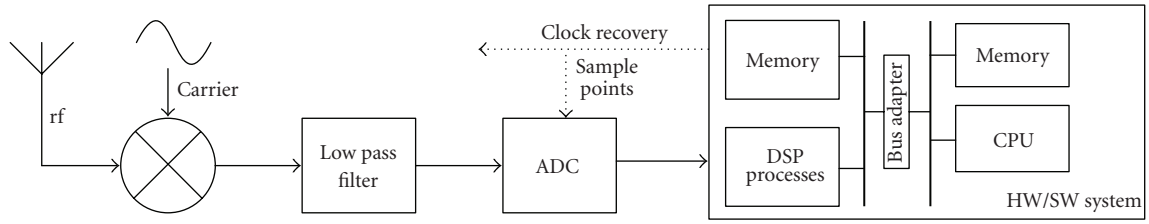

Figure 1: Example for an embedded analog/mixed-signal architecture: SW defined radio.

enough to be accepted and understandable by the different design communities. An executable specification enables a simulation-based process to discover specification flaws and validate the specification before implementing further design steps. A key point for such a methodology is to base it on a language which establishes a common and unambiguous basis in syntactical and semantical terms for specification and simulation tasks. At the same time, the design methodology must support gradual refinement of different parts of the specification. The effect of such refinement steps must be checkable globally, that is, within the framework of the whole system. As it will be seen later, this is related to the support of mixed-level simulation.

SystemC [1] is a system modelling language which is widely accepted and has features to become a common basis for system-level specification and simulation of embedded systems. SystemC is a C++ class library for system modelling and simulation with a discrete event (DE) simulation kernel. SystemC provides additional specification facilities to model time, concurrency, hierarchical partitioning, and hardware constructs like clocks and signal channels. In addition, the language provides facilities for extensibility. This has been recently exploited by several methodologies, based on extension libraries, like SystemC-AMS [2] and HetSC [3]. These libraries support additional MoCs in SystemC, which provide a system-level support for different application domains in a single system-level specification of a heterogeneous system.

Using these libraries enables, for instance, a faster simulation of analogue hardware and signal processing parts and a safer development of concurrent embedded software. Other SystemC extensions, like transaction-level modelling (TLM) [4], are focusing on the application of MoC concepts to support early and fast hardware platform modelling.

Figure 1 shows a software-defined radio (SDR) system as an example of an embedded mixed-signal application. Such a system samples the radio frequency input with a large sampling frequency. The signal demodulation then can be carried out by the digital hardware part with software.

The modelling and simulation of such a system give rise to using a number of different MoCs. Obviously, there is an analogue part to the left of the A/D converter (ADC), and a digital part, namely, the hardware/software (HW/SW) system. But the choice of MoCs for modelling within these two domains might depend on simulation requirements or the abstraction level of the different system parts. For example, the analogue part may be modelled with a signal flow-oriented MoC (like the SystemC-AMS-timed synchronous dataflow (T-SDF) MoC [5]), while the *functionality* of the whole HW/SW system may be modelled using the Kahn process network (KPN) MoC first, abstracting its structure as indicated in Figure 1.

Later refinement steps might require an electrical network model for the low-pass filter, while for the HW/SW system, a structural model is provided with the bus communication abstracted using TLM concepts. A subsequent refinement step then uses a clocked synchronous model for bus communication, while the digital DSP part may be modelled using the T-SDF MoC for simulation speed purposes. This example indicates that the usage of various MoCs in modern system design is natural, and indeed inevitable if a profound theoretical basis is desired. This, in turn, implies the need for conversion means between system parts being modelled using different MoCs. These conversion means should be easy to handle.

1.1. MoC-conversion

While SystemC extensions enable the designer to model and simulate heterogeneous systems with custom MoCs for distinct system parts, their usage also requires conversion means between system parts modelled using different MoCs. These conversion means have to adapt time, communication, and computation domains of the respective system parts, with time adaptation being of capital importance. By abuse of language, we will refer to this conversion process as *MoC-conversion*, in accordance with [6], although one might argue that terms like *MoC-bridging* or *MoC-composition* are more appropriate.

In some cases, the required conversion is natural and explicit, for example, when considering a system consisting of digital and analogue hardwares. Such a system will contain A/D- and/or D/A-converters from the specification level down to the implementation level. Concepts like *mixed-level simulation* [7] give rise to more artificial MoC conversion problems. The idea of mixed-level simulation is to simulate systems such that different parts are specified at different levels of abstraction. The benefit of this approach is the combination of the speed of the system-level simulation with the accuracy of simulation on more detailed levels. The drawback is that every time a system part is refined to a lower abstraction level, the respective MoC will often change

too, while the other parts remain at a higher abstraction level with their MoC unchanged. Therefore, MoC converters are needed here for the sole purpose of modelling and simulation.

MoC converters are usually "static" in the sense that they perform a single adaptation, for example, $MoC_1 \rightarrow MoC_2$. For example, SystemC-AMS provides converter ports to connect T-SDF-modules to discrete event signals. However, the support provided by these static converters can be improved to speed up refinement steps and design space exploration. Each refinement step in a mixed-level design approach may involve a different static MoC converter. This can require a systematic, error-prone, and time-consuming replacement of such static converters.

This article presents an approach to automate the usage of MoC (and also data type) conversion with a SystemC channel called *converter channel*. Converter channels detect the MoCs of the system parts they are connected to via the interface types of the respective ports, and provide automatically the conversion means needed. If the conversion semantic is not obvious, options are provided for the designer to steer the behaviour of the converter channel in certain cases. Moreover, since the conversion is hidden within the channel, it does not interfere with the reuse of partial design blocks.

The rest of the article is organized as follows: Section 2 gives an overview of the related work. Section 3 introduces the two SystemC libraries which the converter channels are based on, namely, HetSC and SystemC-AMS. In Section 4, we show how a converter channel is used. Section 5 treats the internal structure of converter channels, and gives details on how two specific MoC conversion cases are handled. In Section 6, we give an application example. We conclude in Section 7.

2. RELATED WORK

This article focusses on the improvement of MoC connections, a key issue in heterogeneous specification. The work done on heterogeneous specification will be reviewed in the following sections. It will situate this work and will also show its basis.

2.1. Metamodels

Metamodels are an important part of the theoretical study of MoCs. They provide general and formal ways to study and compare different MoCs. This is key to understand the properties of each MoC, as well as the interactions which arise when two or more MoCs are combined in the same specification.

In [8], any concurrent specification is abstracted as a set of *processes* connected by means of *signals*. A *signal* is understood as a set of *events*, while an *event* is a value-tag pair. Neither an order relationship is assumed among the events (although it can be established as a function of the event tags), nor is there any additional implication in terms of communication semantic. Finally, a *process* is defined as a relationship among input and output *signals* of the *process*. This view of a process is quite different from that of a process being a sequence of statements able to change a set of state variables related to it.

ForSyDe (formal system design) [9] is a metamodel also based on *process*, *signal*, and *event* concepts. However, it presents slight differences with respect to the metamodel in [8]. A signal is a sequence of *events*, where an *event* is just a value. Despite there is no tag associated to a ForSyDe event, the definition of a signal as a sequence provides a notion of strict order among the events belonging to it. In addition, ForSyDe defines three types of events, which provide the capability to represent untimed, synchronous, and timed models. A major difference to [8] is that a ForSyDe process is defined in an introspective way, by means of process constructors, which take as input the functions which relate the input and output signals. A strength of ForSyDe is a set of transformation rules, which enable a safe formal procedure for the MoC refinement process involved in the design process, when the specification is refined into the implementation.

Although metamodels provide a formal basis for a heterogeneous specification methodology, a metamodel is not a methodology itself. ForSyDe, however, is also based on a Haskell library [10], such that ForSyDe models can actually be implemented and simulated. A handicap is that Haskell is not quite extended among the design community.

There are several approaches to provide heterogeneous specification methodologies. Two main approaches can be distinguished. On one hand, the extension of hardware description languages (HDLs), mainly for analogue extensions, on the other hand, the extension of high-level languages like UML, Java, or C++. These proposals will be reviewed in the following sections.

2.2. Extension of HDLs: Verilog-AMS and VHDL-AMS

The mixed signal hardware description language Verilog-AMS [11] incorporates two different MoCs for simulating analogue and digital hardware, respectively. While digital simulation is handled by a DE simulation kernel, a differential equation kernel is engaged with the simulation in the analogue domain. *Disciplines* can be defined to be associated to these domains, for example, an analogue discipline for electrical signals or digital disciplines for binary or multivalued logic. To connect modules belonging to different disciplines, Verilog-AMS provides the features of *connect modules* and *connect modes* [12]. Connect modules must be implemented by the designer and define the interaction between modules of different disciplines. Apart from signal conversion, connect modules can also model subtle behaviour like the influence of a capacitor to the delay of a digital signal. With connect modes, the designer steers the automatic insertion of these connect modules.

The VHDL-AMS language [13] is an extension of the IEEE 1076 (VHDL) standard that supports description and simulation of analogue, digital, and mixed-signal circuits and systems. A basic principle is not to rely on a specific algorithm, but on a mathematical foundation to solve the implicit or explicit differential algebraic equations (DAE)

which describe the analogue part of the system specification. Models must not depend on the time steps taken by the analogue solver. VHDL-AMS adds two new object types to VHDL: the *quantity* and the *terminal*. Both can either be a local or an interface object, and they are used in conjunction with basic VHDL primitives, like *entity* and *architecture*. Specifically, input and output ports can be declared either as terminals or as quantities. Quantities are floating point scalars, which are the unknowns of the DAEs solved by VHDL-AMS. Quantity associations create a signal flow model. Terminal associations of the same *nature* can be used to create an analogue net list. A nature restricts the association of terminals to those of the same nature. A nature also relates *across aspects*, which are quantities which represent effort like effects (voltage, pressure, etc.), with *through aspects*, which represent flow-like effects (current, fluid flow rate, etc.). An example of nature is the *electrical nature*, which relates voltage quantity with current quantity. These basic elements let the user specify implicit DAEs. A set of explicit concurrent statements enables the specification of explicit DAEs. Among them, the simultaneous procedural statement allows the formulation of the DAE as inline sequential code.

Analogue extensions of HDLs provide support for the specification of mixed signal systems, combining digital, and analogue domains. However, they lack support for more abstract MoCs, and are able to consider abstract models in the early design phases and for embedded software, an important part of embedded systems. We now review specification frameworks based on extensions of high-level languages, which cover these deficiencies better.

2.3. Extensions of high-level languages

The lack of a unified system specification language is one of the main obstacles bedevilling SoC designers [14]. A common specification language is a major aid in generating a heterogeneous specification methodology. It states a common and unambiguous syntax and semantic for the specification facilities. This improves the understanding between the different design areas and enables exchanging the unambiguous models, independent of the tools used by each part for graphical capture, verification, performance analysis, synthesis, that is. Therefore, extensions of programming languages like UML, Java, or C++ have been proposed for system modelling and simulation.

2.3.1. Ptolemy II

Ptolemy II [15] (the latest evolution of the Ptolemy project) is one of the first and most important approaches to provide a unified framework for the specification and simulation of heterogeneous systems. A Ptolemy specification consists of a set of components called *actors*. *Actors* communicate among them by means of *ports*. *Actors* can be either composed out of a set of other actors or primitive. In the latter, the actor has a specific functionality described in Java. The Java functionality is distributed in a set of internal methods (*initialise*, *prefire*, *fire*, *postfire*, etc.), which are callbacks that execute functional code and perform read and write accesses to the actor ports at different times of the simulation. These callbacks are governed by a Ptolemy *director*. A *director* is associated to an *actor* and defines its *domain*, that is, the MoC. The callback structure and a graphical capture environment called *Vergil* provides a kind of common specification framework to Ptolemy, while Java has more an implementation role.

Heterogeneity is handled through hierarchisation, in the sense that actors on the same hierarchy level obey the same *director*, thus the same MoC. Each specification instantiates its own hierarchy of directors, which coordinates actor executions. This helps MoC connection, since the order and time position in which each token is transferred across the domains is explicitly controlled by the directors' hierarchy. Because of this, in general, no special MoC interface infrastructure to connect actors of different domains is necessary, apart from port communication. In addition, Ptolemy introduces the *ModalModel* for models which let specify different modes of operation for a model under any of the supported MoCs. By means of a discrete FSM, the different mode transitions which the model can traverse is specified. Each mode has a *refinement*, under the specific model embedded in this mode FSM. Then, *Hibrid Systems* embed continuous time models in a mode FSM. In any case, Ptolemy provides a hierarchical approach to heterogeneity.

2.3.2. SysML

SysML (systems modelling language) [16] is a domain-specific modelling language for system engineering applications. It supports the specification of systems which may include hardware, software, information, processes, personnel, and facilities. The main idea is that UML is well suited for software modelling, but too comprehensive and imprecise for domain-specific applications which require nonsoftware components. Therefore, SysML is constituted as a *profile* which reuses a subset of UML 2.0. At the same time, SysML also extends UML 2.0 providing diagrams for the capture of requirements and parametric constraints. These capabilities enable requirements engineering and performance analysis, two major activities in system design.

SysML was originally developed as an open source specification project. In 2005, the OMG (bject management group) derived the OMG SysML [17], which is defined as SysML, highlighting it as a graphical language. Graphical capabilities of SysML for the specification capture are endorsed by a wide set of modelling tools supporting UML 2.x and the SysML profile. Thus, SysML provides a common, rich, and graphical method to model systems under specific design domains. However, SysML, as other UML profiles must rely on an implementation language to be executable or simulatable. For instance, in [18], SysML is used for the specification of analogue systems. That is, SysML is used to model the dynamics of continuous systems by means of DAEs. However, since UML lacks analogue solvers, the solution adopted is to translate SysML constructs to Modelica language constructs.

2.3.3. Metropolis

Metropolis [19] is a specification methodology which is (like Ptolemy) based on Java. It also targets simulation, verification, and synthesis (mainly of software). Metropolis supports heterogeneity in two levels. In the lower level, it defines a metamodel language which provides a class-based infrastructure. The classes have a well-defined semantic, but are general enough to support existing MoCs and new ones layered in an upper level.

In Metropolis, there are basic specification elements, like the *process*, the *port*, and the *interface*, which are similar to other specification methodologies like SystemC. *Processes* are atomic computation elements, defined as sequences of events mapped to threads. The *port* is the unique mean of a process to communicate with other processes. *Interfaces* are a set of communication methods. Other specification facilities, for example, *media*, *quantity manager*, and *state media*, are specific of Metropolis. *Media* are the primitives for communicating processes. They connect processes through *ports*. *Quantity managers* fix constraints whether processes must be scheduled. *Quantity managers* communicate among themselves by means of *state media*. The metamodelling character of the lower level of Metropolis is confirmed by the ability to combine executive parts with declarative parts. Declarative parts enable the specification of nonfunctional constraints.

Metropolis is able to support heterogeneous design building specific models over a general framework. This is mainly achieved by means of the different implementations of *media*, whose semantic mostly defines the MoCs employed. This is a major distinction with respect to Ptolemy, where MoC semantic mostly relies on the *director* class and where *actors* connections play a more secondary role. Strengths of Metropolis are its high expressivity and its formal framework, which enables the connection of synthesis and verification tools.

Regarding MoC connection, the Metropolis metamodel defines a core element called *adapter* or *wrapper*. It is modelled as a process in charge of adapting the process which contains the actual functionality to the medium this *functional* process is connected to. This adaptor determines the firing condition of the functional process and the interaction with the connected media. Therefore, when processes are refined and different MoC connections are involved, this requires only changing the adapter (the functional processes are not affected). In addition, *Quantity managers* can somehow be employed to control the execution semantic of the different components of the specification, thus to ensure a coherent coordination of their simulation, as it is done in Ptolemy by means of *directors*.

2.4. SystemC extensions for heterogeneous specification

Several reasons lead research community to propose C++ based extensions for heterogeneous system-level specification, which make use of the object-orientated features of C++. Executable specifications using such extensions are compilable and do not rely on a virtual machine, which can lower the simulation performance, a key aspect in system-level design activities. In addition, C++ syntax is familiar to many embedded system and HDL designers.

Talking about C++ extensions for system-level specifications has become almost synonymous to talking about SystemC extensions. SystemC is an IEEE standard language that has started to play a role as unifying system-level language for embedded system design. The SystemC language reference manual [20] states a syntax and an unambiguous semantic for the language constructs, which is a strong basis for SystemC-based methodologies supporting heterogeneous specification.

2.4.1. SystemC-H

SystemC-H [21] 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 supported MoC. The current scope of the SystemC-H library covers the SDF and communicating sequential processes (CSPs) untimed MoCs. For instance, SystemC-H provides a solver for static scheduling of SDF graphs which enables schedulability analysis and provides a 75% speedup with respect to the DE MoC [21]. Another result of this work is that the addition of a specific solver is not always worthwhile, since the simulation speedup for some abstract MoCs can be negligible [22]. In addition, similar speedups were reported for the dynamic approach to SDF for large-grain SDF specifications [3]. Finally, the combination of MoCs, can also spoil the speedup gained by using a specific MoC. For instance, the speedup decreases to 13% for a mixed DE-SDF example [21]. A major drawback of SystemC-H is that it is implemented by modifying the SystemC standard library.

2.4.2. Analogue extensions of SystemC

SystemC-A [23] proposes a general approach to provide analogue extensions in SystemC which is able to handle a wide range of nonlinear dynamic systems. SystemC-A is a superset of SystemC which provides support for several abstraction levels, with a special focus on a higher abstraction level. However, it also supports circuit-level descriptions. SystemC-A proposes a general-purpose analogue solver coupled with the DE kernel of SystemC by means of pessimistic (lock-step) synchronisation. However, this synchronisation requires the modification of the SystemC kernel.

The SystemC-WMS [24] library provides basic electronic components and blocks for analogue macromodelling, such that the specification of analogue models can combine parts at low- and high-abstraction levels. Analogue blocks can communicate by exchanging energy waves using *wavechannel* interfaces. Thus the methodology enables the definition of a standard analogue interface. In [24], it is also claimed that SystemC-WMS can support simple types of nonlinear DAEs.

HetSC and SystemC-AMS are proposals for extending SystemC for heterogeneous specification which are directly

involved in our work. They will be reviewed in detail in the following chapter.

3. HetSC AND SystemC-AMS

The task of using HetSC together with SystemC-AMS for system modelling and simulation came up due to the project ANDRES (**AN**alysis and **D**esign of runtime **RE**configurable heterogeneous **S**ystems [25]). ANDRES explores design methodologies for heterogeneous systems which are able to adapt themselves during runtime to changing user requirements as well as varying environmental constraints. The project encompasses the development of

(i) a theoretical framework for the specification of such adaptive heterogeneous systems (AHESs);

(ii) a SystemC-based modelling framework where the different domains are represented by specific MoCs;

(iii) means to automatically synthesise the components and interfaces of AHES regarding the digital hardware and the software.

The work presented here is an integral part of ANDRES. While the key concern of ANDRES is the consideration of adaptivity in all the domains involved, it is obvious that the integration of the MoCs and methodologies involved in the project is essential. The SystemC extensions used are the following.

(i) *SystemC-AMS* [2] targets modelling and simulation of analogue and signal processing systems. It provides the MoCs (timed) SDF and continuous time (with respect to linear electrical networks (CT-NET)).

(ii) *HetSC* [3] is intended to model systems using different MoCs (e.g., Kahn process networks (KPN) and synchronous reactive (SR) systems), and also provides an entry point for software synthesis.

(iii) *OSSS+R* [26] is a library for object-oriented modelling of run-time reconfigurable hardware, which will provide direct hardware synthesis capabilities targeting dynamically reconfigurable FPGAs.

The difficulty lies in getting the three libraries to work together with respect to the different MoCs involved. Since the modules in OSSS+R are basically clocked synchronous, which is a MoC provided by SystemC directly via its discrete event (DE) kernel, this is a challenge which mainly concerns HetSC and SystemC-AMS.

3.1. HetSC

HetSC [3] is a methodology for enabling heterogeneous specifications of complex embedded systems in SystemC. MoCs supported include untimed MoCs, such as KPN, its bounded FIFO version bounded Kahn process networks (BKPNs), communicating sequential processes (CSPs), and synchronous dataflow (SDF). Synchronous MoCs, such as synchronous reactive (SR), clocked synchronous (CS), and

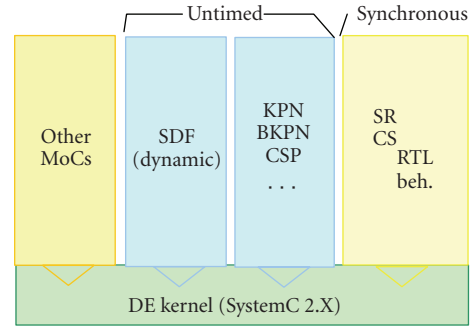

Figure 2: HetSC MoCs over the DE strict-time kernel of SystemC.

the timed MoCs already supported in SystemC are also included (see Figure 2). HetSC aims at a complete system-level HW/SW codesign flow. Indeed, the methodology has been used together with other system-level profiling and software generation methodologies [27].

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 the 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. In [3], a simulation speedup similar to that of kernel extension was reported for large grain specifications. It was argued too that once analogue parts are connected to the model, usually more critical in terms of simulation time, the consideration of Amdahl's law, can make almost undistinguishable the differences between the speedups got by the different implementations of abstract MoCs, independently on whether such implementations extends the kernel, like in [21], or to the contrary, it is layered over the strict-time DE kernel of SystemC, like HetSC does. HetSC provides the strategical advantage of not requiring the modification of the standard SystemC kernel, since the new features are provided by the HetSC library, what makes it a more decoupled approach.

3.2. SystemC-AMS

SystemC-AMS [2] is a specification methodology developed by the open SystemC Initiative (OSCI) SystemC-AMS working group. The main goal is to extend the HW/SW-oriented SystemC towards a framework that supports

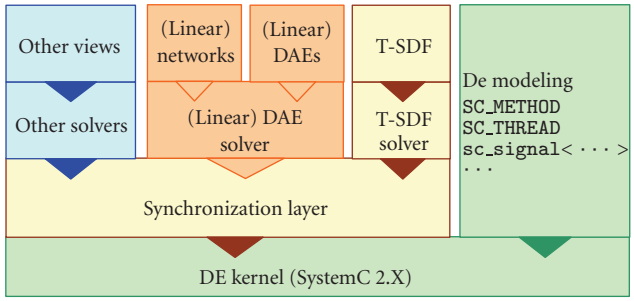

Figure 3: SystemC-AMS layers over the strict time DE kernel of SystemC.

functional modelling, architecture exploration, system integration, verification and virtual prototyping of embedded Aanalogue/mixed-signal (E-AMS) systems. The most important MoCs (in addition to the DE-MoC) required for this purpose are

(a) continuous time electrical networks (CT-NET);

(b) continuous time differential equations/transfer functions (CT);

(c) timed synchronous data flow (T-SDF).

Due to the interaction between the SDF and the other timed MoCs, SystemC-AMS attaches time semantics to SDF by settings fixed time steps to SDF cluster execution times. We therefore refer to the SystemC-AMS SDF MoC as "timed SDF" (T-SDF). For modelling asynchronous systems, time steps can also be triggered by events.

A major challenge is gaining sufficient simulation performance while accurately modelling the architecture's behaviour at the same time. At least for electrical networks, no mature and dependable approaches are available that allow us to stay in the SystemC 2.2 framework without kernel extensions. Simulation of electrical networks requires a structural analysis, setup of equation systems, and numerical methods for solving them. For T-SDF, measurements indicate that the use of extended kernel capabilities provides a factor 4 speedup for DSP functions frequently found in E-AMS such as FIR filters [28]. Therefore, SystemC-AMS provides kernel extensions for CT-NET, CT, and also T-SDF simulation.

The benefit of using T-SDF regarding simulation speed is twofold. Since it is SDF, it is possible to compute a static schedule for the T-SDF processes before the simulation starts (like in the current SystemC-AMS prototype), such that the scheduling overhead during the simulation is minimised. Additionally, using large data rates leads to schedules where certain processes might be computed repeatedly in a row, such that the number of context switches reduces also.

To synchronise the kernel extensions with the SystemC DE-kernel, SystemC-AMS uses T-SDF with discrete time steps and/or controlled time steps. This has the advantage that all modules are executed in signal flow's direction. Partitions of a model, simulated by one kernel extension, are encapsulated in a SDF module. The partition communicates with other components that use other kernel extensions only via directed (T-SDF) signals. For convenience, however, electrical networks can also interface DE directly.

The implementation of SystemC-AMS is organised in three layers. The top layer (view layer) includes the classes visible directly by a designer, and is used directly for specifying models. It provides, for example, an SDF-module class together with SDF-signals and port. The middle solver layer has kernel extensions that are used by the view layer. To couple different solvers, SDF with constant time and with user defined synchronisation events is used in the solver layer. Figure 3 shows the overall organisation of the SystemC-AMS extensions.

3.3. Connecting HetSC and SystemC-AMS

In [29] the connection of SystemC-AMS and HetSC was explored. It was shown, that both methodologies can collaborate to support a wide spectrum of MoCs. Moreover, the collaboration of these methodologies provides an efficient balance between MoCs directly supported over the DE strict-time kernel, and MoCs relying on additional synchronisation and solver layers. The idea is that specific solvers are provided only for a set of MoCs where the simulation speed up is significant. In this approach, this set corresponds to analogue MoCs where the simulation speed ups can be of several orders of magnitude, while 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. Thus, in these cases, the static scheduling provided by the T-SDF solver of SystemC-AMS should be favoured.

In [29] the ways in which HetSC and SystemC-AMS enable MoC connection was also explored. HetSC provides the concept of *border processes* and *border channels* for the connection of the various MoCs. In this way, common SystemC elements are employed in MoC interfaces. Border processes are similar to Metropolis adapters in that it requires from the user to explicitly write the adaptation code. In contrast, border channels hide this adaptation code. Border channels make a syntactical adaptation. For instance, on one side they can offer a FIFO like interface, while on the other side, they offer a rendezvous-like interface. Moreover, border channels can also adapt different semantics of connected MoCs. HetSC border channels mainly focus on the adaptations performed in the time domain. SystemC-AMS also provides similar ways for MoC connection. It provides converter ports and facilities to enable communication between different MoCs (i.e., DE with T-SDF, T-SDF with CT-NET, etc.). In other cases, MoCs, such as transfer function models, are actually embedded in T-SDF modules.

For SystemC-AMS, *polymorphic signals* [30] were developed to connect modules modelled under different MoCs (CT-NET, T-SDF or SystemC-DE). Like the HetSC border channels, they have to do a syntactical and semantical adaptation. An important benefit of polymorphic signals is that they are able to select the right MoC connection automatically. This is done before the simulation starts

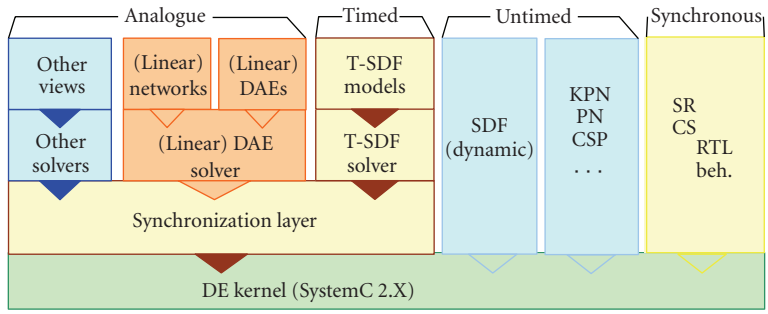

Figure 4: Integration of SystemC-AMS and HetSC.

(at *elaboration time*). The polymorphic signal detects the MoCs of the modules it is connected to and provides the appropriate conversion means without the designer's assistance who only sets different options.

4. CONVERTER CHANNELS

Converter channels, first introduced in [29], are meant to unify the approaches of polymorphic signals and border channels, while providing adaptation in the data type domain. Converter channels can connect modules specified using different MoCs, and automatically adapt their different handling in the semantic of time, communication, and data types. In this way, converter channels provide an advanced facility for the automatic syntactical and semantical connection in heterogeneous specification, refinement and design exploration methodologies based on SystemC.

Before we describe how a converter channel works, we demonstrate its use, that is, how they appear to the designer. The internal structure and conversion semantics will be covered in the next chapter. A converter channel is instantiated with up to six template parameters:

```
converterchannel < DT_W, DT_R1,...,DT_R5 > name;
```
(1)

the first parameter, `DT_W`, is mandatory and denotes the data type of the writing port the signal is going to be attached to; it can also be used as a data type of a reading port. The parameters `DT_R1` to `DT_R5` are optional and denote additional data types on the reading side.

The choice of the number of additional reading side data types to be five is somewhat arbitrary, but since the use of an unlimited number of template parameters in C++ is not possible, a number had to be fixed. It is reasonable to assume five additional data types to be more than sufficient for any reasonable application. However, if the compilation overhead proves to be to large, we might reduce this number.

The connection of the converter channel to module ports has to be done by named mapping and works like in Algorithm 1, where a T-SDF-module producing double values is connected to a T-SDF-module reading integer values and an ordinary SystemC-module (DE) reading double values (see also Figure 5).

As Algorithm 1 shows, the usage of a converter channel differs a little from the usage of an `sc_signal`

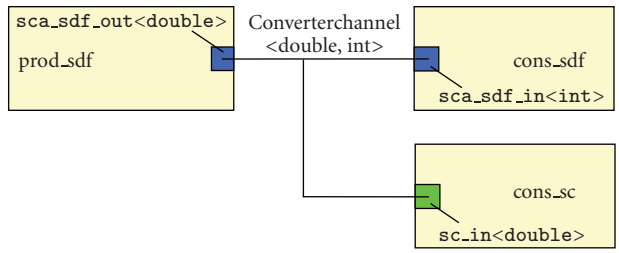

Figure 5: Illustration for Algorithm 1.

```
converterchannel < double, int > sig;

producer_sdf_double prod_sdf ("prodsdf");
prod_sdf.out (sig.source_sdf());

consumer_sc_double cons_sc("conssc");
cons_sc.in (sig);

consumer_sdf_int cons_sdf ("conssdf");
cons_sdf.in (sig);
```

Algorithm 1: Connecting a converter channel.

regarding the connection to the writing port, which contains a declaration of the MoC used on the writing side: instead of `prodsdf.out (sig)`, the code line `prodsdf.out (sig.source_MOC())` has to be used, where MOC denotes the MoC on the writing side. In contrast, the connection to ports of reading modules works as usual; in particular there is no need to declare the MoC of the reading module.

The need for MoC-declaration on the writing side has technical reasons. In principal, MoC and sense of the signal flow of a port (i.e. in- or output port) can be recognised automatically by the port's interface type. Unfortunately, for the `sca_sdf_port <>` class of SystemC-AMS, both the in- and output ports happen to implement the same interface. Therefore, we can recognise the MoC here, but not the signal flow sense. Since there is only one writer for each converter channel, using a MoC-specific binding method for the writing side has the least coding overhead. Additionally,

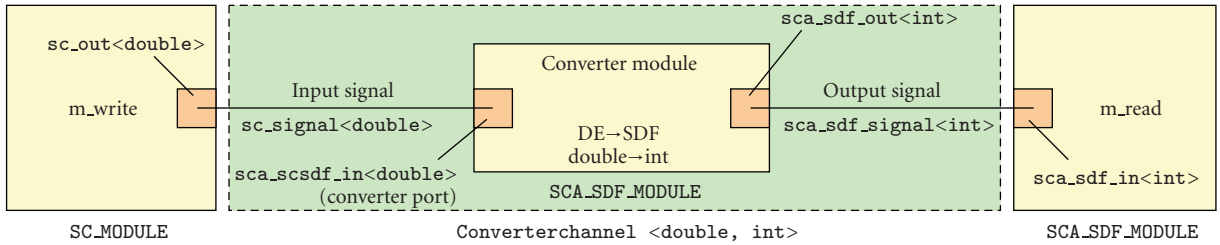

Figure 6: Example of the internal structure of a converter channel.

this approach offers the opportunity to pass additional MoC specific parameters to the converter channel using the binding method.

The designer has to make manual changes at a converter channel only if

(i) the MoC of the writing module is changed; then the connection of the converter channel to the port has to be changed, for example from

```
module.out (signal.source_sdf ())
```
(2)

to

```
module.out(signal.source_sc ())
```
(3)

(`sc` represents the standard SystemC DE MoC).

(ii) the data type of a port of a reading module is changed and is not already a template parameter in the instantiation of the converter channel; then it has to be added. So, for example, the code of the instantiation could change to

```
converterchannel < double, int, sc_bv < 8 >> sig;
```
(4)

furthermore, it is possible to set options on the converter channel, for example, to control the data type conversion semantics. If, for example, the writing side of a converter channel is of type `double`, with a `sc_uint < N >` port on the reading side, this situation has the potential for information loss if `N` is too small or if the writing side provides negative values. For such a case, the converter channel offers a method

```
sig.setRangeScaling (min, max)
```
(5)

to adapt the value range of the writing side to those of the reading side. For applying this function, the designer has to know (or has to have at least an idea of) the range of the input signal. Therefore, if the values of the writing side turn out to exceed the interval [min, max] during the simulation, warnings are produced.

This example also illustrates the potential risks of the automated data type conversion capabilities of a converter channel. We will not, however, go into detail on data type conversion issues in this article. More on this can be found in [31].

5. INTERNAL STRUCTURE AND OPERATION

In this section, we provide details on how the code described above is processed, that is, handled internally in terms of signals and converter modules. Let us assume that a converter channel connects two modules `m_write` and `m_read` having a port out and in respectively. Four cases can occur:

(1) The MoCs of the modules as well as the data types of the ports mismatch.

(2) The MoCs of the modules are different, but the data types are the same.

(3) The MoCs of the modules match, but the data types of the ports do not.

(4) Both MoCs and port data types match.

In case 1, the converter channel instantiates two appropriate signals (the input and the output signal), which then are connected to the ports of `m_write` and `m_read` respectively. These, in turn, are connected to an appropriate converter module. An example is shown in Figure 6: `m_write` is a SystemC DE module having a port `out` of type `sc_out < double >` and `m_read` is a SystemC-AMS T-SDF module having a port `in` of type `sca_sdf_in < int >`.

To achieve conversion between SystemC-DE modules and SystemC-AMS T-SDF modules, the SystemC-AMS library provides `sca_scsdf_in` and `sca_scsdf_out` ports, which can be used within T-SDF-modules to connect to DE-signals (i.e., `sc_signals`). Therefore, a converter module from DE to T-SDF can be realised by a T-SDF-module having an `sca_scsdf_in` input and an `sca_sdf_out` output. Then it can be connected to `m_write` and `m_read` with an `sc_signal` and `sca_sdf_signal` respectively. We will discuss the related conversion semantics shortly.

After reading the input signal, the value is converted to the desired target data type and written to the output of the converter module. The data type conversion function here is inherited by a special data type conversion class which has templated specialisations for each data type pair. Hence, the conversion modules are implemented using two separate levels: MoC conversion and data type conversion.

In case 2, the procedure is similar; there is no need for data type conversion, and the converter module just passes the value of the input signal to the output.

Case 3 is also fairly similar, with the difference that all ports and signals used are of the same type with respect to

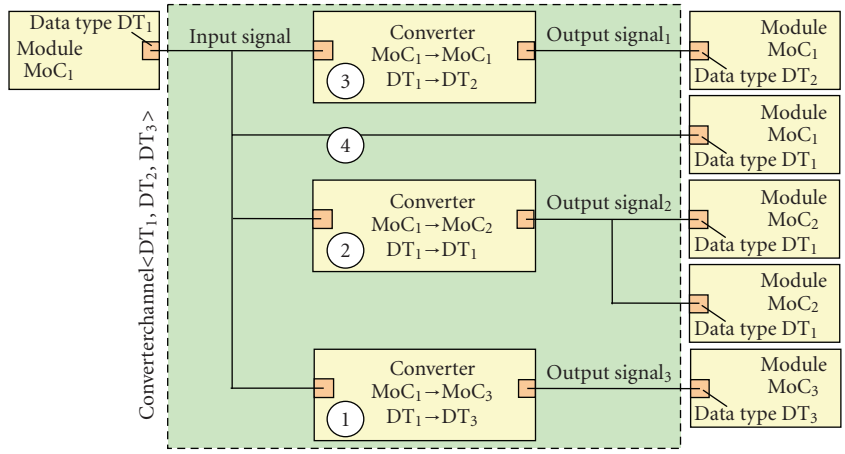

Figure 7: Internal structure of a converter channel connected to multiple reading modules.

the MoC; in particular the converter module belongs to the same MoC as the two modules to be connected. As in case 1, the input signal is read and the converted value is written to the internal output signal.

Obviously, the simplest case is case 4, where there is no need for any conversion at all. Here, the converter channel simply generates only the appropriate input signal, and connects it to `m_write` and `m_read` directly without a converter module in between.

In general, a converter channel will be connected to several readers. Here, apart from the input signal, several converters and output signals are created. Each converter module is then connected to the input signal and to its own output signal. Each reading module is connected to the appropriate output signal. If a reading module happens to agree to the writing module regarding MoC and data type, it is directly connected to the input signal. Figure 7 shows an example (the numbers refer to the four cases stated above). The capability of the converter channel to have readers with several data types is also an extension of the concept of polymorphic signals [30], where data type conversion was also included, but the reading modules had to share the same data type.

In the next section, we show how MoC conversion is handled in the case of DE↔T-SDF and KPN/BKPN↔T-SDF conversion.

5.1. DE↔T-SDF conversion

The conversion between T-SDF of SystemC-AMS and the SystemC DE MoC is basically straightforward, due to the strict time semantics of T-SDF. SystemC-AMS already provides conversion facilities for this, which are used by the converter channels, while also resolving potential semantical issues.

In the T-SDF→DE case, the converter channel instantiates an internal converter module which is connected to the writing side via an `sca_sdf_signal < DT_W >`, and which makes use of the SystemC-AMS `sca_scsdf_out <>` converter port to connect to the reading side via an internal `sc_signal < T >`.

A token which is written to the internal `sca_sdf_signal < DT_W >` by the writing side at SystemC-AMS time t is converted such that the internal `sc_signal < T >` holds the value of that token from SystemC time t on. Note that there is a value change event on the reading side *only* if there is an actual value change of one token to the next one, which might be a semantical problem in certain models, when the DE side is supposed to read *every* token. For these cases, the converter channel offers a clock signal running with the pace of the T-SDF side, which can be accessed using the method `sdf_clock ()`.

With respect to the previous case, the DE→T-SDF case is exactly mirrored regarding the use of internal signals and converter ports (see Figure 6). The conversion semantics is such that a value which is written to the internal `sc_signal < DT_W >` by the writing side constitutes a *current* value for the signal. This current value is then sampled by the T-SDF side with a frequency determined by its sampling period.

There is, however, a potential for loss of information. If the DE side changes the value twice or more in between two sampling instances by the T-SDF side, these interim values will be lost. Therefore, the converter channel provides warnings in this case with the help an internal DE-module which is sensitive to the value changes of the internal `sc_signal < DT_W >`. At each sampling instance, the internal converter module asks this module how many value changes occurred since the last sampling instance. If this value exceeds one, it raises a warning or a runtime error respectively, depending on the designer's choice. This behaviour can also be switched off completely.

5.2. KPN/BKPN↔T-SDF conversion

In [32], several types of MoC connections were distinguished with regard to the time domain. The basic observation is that when two MoCs are connected which handle time at a

different detail level, there is an effect of time information injection from the more detailed MoC to the less detailed MoC. The way by which time is handled is often related to other aspects associated to communication semantics. For instance, it is common to associate consuming reads and nondestructive writes to untimed MoCs, while destructive write and nonconsuming reads semantic of HDL signals (e.g., the `sc_signal` channel semantic in SystemC) is suitable for timed-clocked MoCs.

In previous works, semantical issues dealing with untimed-untimed [32] and untimed-synchronous [33] MoC connection have been addressed. We will consider a first approach to the untimed-timed and the timed-timed MoC connections, where the untimed-timed case refers to the connection of untimed Kahn process network (KPN) and bounded Kahn process network (BKPN) MoCs of HetSC with the timed synchronous dataflow MoC (T-SDF) of SystemC-AMS. The timed approach of T-SDF can actually be considered as a discrete time MoC, since each cluster execution has a specific SystemC time stamp. The timed-timed case occurs when the addition of of strict-time information to the KPN/BKPN network connected to the T-SDF part has to be considered. We have four cases to consider:

(1) KPN/BKPN→T-SDF;

(2) T-KPN/T-BKPN→T-SDF;

(3) T-SDF→KPN/BKPN;

(4) T-SDF→T-KPN/T-BKPN.

For each case, we give the semantic of the converter channel for the writing and the reading sides. This includes the problem of time information injection for the timed-untimed connections. If the overlap of untimed/timed write and untimed/timed read semantics generate any inconsistency, the converter channel has to perform appropriate actions.

We start with the KPN/BKPN→T-SDF connection. From the writing side, the KPN/BKPN characteristics of the untimed network does not carry semantical problems. The converter channel can be either configured as an unbounded FIFO (as the HetSC `uc_inf_fifo` channel), such that the untimed part will never block. Or it can be configured as a bounded FIFO (as the HetSC `uc_fifo` channel), such that the untimed part can block. In any case, there is no loss of data. Syntactically, this is achieved by connecting the port `out` of a writing module `module` with the code line

```
module.out(sig.source_fifo (size));        (6)
```

to a converter channel `sig`. If the integer parameter `size` is omitted, the internal buffer of the converter channel will be unbounded.

The time-domain and communication-domain adaptations are closely related. The KPN/BKPN not only expects to do a nondestructive write, but also a destructive read which frees space in the internal buffer. Therefore, data consumption is needed on the reading side. This might appear like a contradiction, since, from the T-SDF side, the read is nonconsuming (as well as nonblocking). However, the read is nonconsuming only for the reads done at the same time stamp (t). From a sampling time stamp (t) to the next sampling time stamp ($t+T$), after a sampling period T, a data token is deallocated and frees space in the internal buffer, establishing a semantical coherence.

On the T-SDF reading side, a token is indeed consumed, but can be read as many times as wanted, during a sampling period T. That way, time information is injected in the untimed part. If, for instance, the converter channel has internal buffer size 2, and the sampling period is 1 millisecond, then the original untimed part will be able to initially write up to 2 tokens and compute until the point when it tries to write the third token. Then, it has to wait for 1 millisecond.

However, corner conditions can still lead to inconsistencies. In the KPN/BKPN→T-SDF case, the inconsistency comes when the internal buffer is empty and the read access of the T-SDF part has to consume a token. The converter channel offers several options for this case.

(i) *Error:* an error is raised and the simulation is stopped.

(ii) *Constant:* a prefixed value is returned. This value remains constant during the simulation. A default value is defined for each data type (i.e., `false` for `Bool` or `SC_LOGIC_X` for `sc_logic`), but can be overwritten by the designer.

(iii) *Hold:* the last value read is returned. It can be seen as a variation of the previous one, where the returned value, instead being constant, is the last token which could be consumed.

In the last two cases, warnings are raised. The designer can choose the desired behaviour by calling the method

```
converterchannel.setFIFO2SDFemptybuffer
(< option >);        (7)
```

where < `option` > stands for either `error`, `constant` or `hold`, where `error` is set by default. This ensures that the designer will be aware of this corner situation resulting from the specification and/or simulation.

The timed-timed T-KPN/T-BKPN→T-SDF connection can be treated basically like the KPN/BKPN→T-SDF case, including the handling of empty internal buffers. In particular, there is no difference regarding the syntax. There is, however, a subtle semantical difference regarding the empty FIFO corner case. In the untimed case, there are basically two causes for empty FIFOs:

(i) The writing process is finite, thus producing only a finite number of token. This will cause an empty FIFO if the simulation time is long enough. Depending on the application, choosing the consume & constant option here can make sense.

(ii) A (partial) deadlock in the KPN/BKPN side involving the process which writes the converter channel. This case is probably treated best with the default consume & error behaviour.

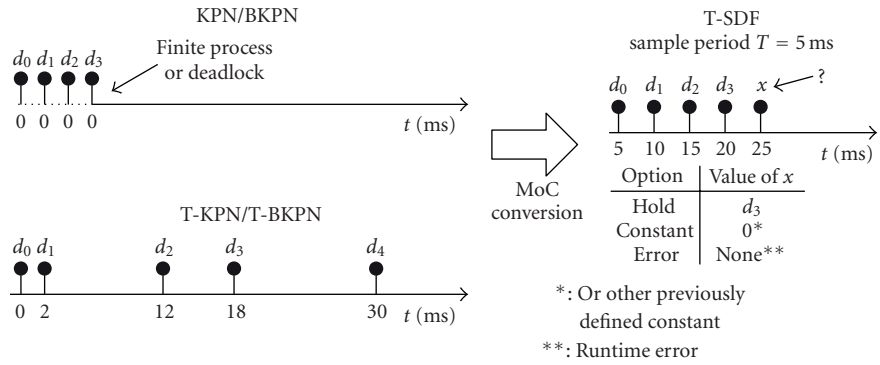

Figure 8: Time-domain behaviour for the KPN→T-SDF and BKPN→T-SDF conversion.

In the timed case, we have a *third cause*, namely the production rate of the writing side being insufficient compared to the consumption throughput of the T-SDF part. In fact, this is a case where the consume & hold behaviour makes perfect sense.

Note that reasons for the KPN/BKPN part to become timed may also be subtle. On one hand processes involved could contain explicit `wait()` statements. On the other hand, there is also the possibility of a timing injection from a third system part communicating with the, originally untimed, KPN/BKPN part. The variety of reasons for empty internal buffers makes it obvious that there is no a-priori intuitive solution to this problem, and the designer has to decide which of the three options offered by the converter channel suits best.

Figure 8 illustrates the conversion process described with an example showing the result depending on the chosen option. The untimed part, either KPN or BKPN, produces the sequence of tokens $\{d_0, d_1, d_2, d_3\}$. The order information, that is, the indexing of the values, is the only relevant time information. For instance, the specification of the untimed part only forces token d_1 to appear before token d_2, but forces nothing else about their SystemC time tags. In Figure 8, a possible time tag assignment consisting in all the tokens having the 0 millisecond time stamp is shown. Note that this is consistent since this order can be reflected by different deltas of the SystemC simulation. After producing d_3, the KPN/BKPN part stops producing tokens.

Assuming that the T-SDF part has a consumption rate 1 with a cluster time of $T = 5$ ms, the converter channel semantic consists in consuming each new token present in the inner buffer of the converter channel at a T pace. Since this pace of T-SDF cluster execution is unstoppable, in this example, a point in time is found (SystemC time tag of 25 seconds) when the T-SDF part needs to consume a data but the inner buffer is empty. In this situation the converter channel exhibits its flexibility. In order to still enable automatic conversion, the converter channel presents the default behaviour, that is, it raises an error. This could coincide with the designers intention, for instance, to detect a deadlock situation in the untimed part.

However, let us assume that the timed part models the sourcing of a kind of irregular pulse signal composed first by the finite subsequence $\{d_0, d_1, d_2, d_3\}$, followed by the infinite subsequence {'0', '0', '0', ...}. In this case, it makes sense to configure the converter channel to adopt the *constant* semantic for the corner case (X = '0'), instead of stopping the simulation and raising an error.

On the left bottom side of Figure 8, a similar case where the KPN/BKPN part suffered a time annotation, becoming a T-KPN/T-BKPN part, is shown. In this case, the empty buffer condition is not caused by a deadlock, but is the result of slow data token generation by the T-KPN/T-BKPN part. Token d_4 is generated at 30 seconds, too late to avoid the corner situation given at $t = 25$ seconds. In a case like this, the *hold* semantic for the corner case ($X = d_3$) is useful to model a sampling and hold behaviour.

Let us now turn to the timed-untimed T-SDF→KPN/BKPN conversion, where we have to introduce the FIFO semantic from the reading side. Here, the T-SDF cluster behaves as a kind of master which transfers tokens to the untimed part at the pace of the sampling time. We decided to use an unbounded internal buffer as the default case, but the designer can also limit its size by calling the method.

```
converterchannel.setReadingFIFOSize(int size).
```
(8)

The modification of the sense of data transfer changes the location of the semantic inconsistency. While the empty buffer does not represent a problem because the KPN/BKPN part remains blocked till the next data arrival, the full buffer condition is problematic, since the T-SDF has to write at its unstoppable pace. This inconsistency only appears when using a bounded internal buffer, and can appear due to several causes, similar to the ones presented for the KPN/BKPN→T-SDF case and the T-KPN/T-BKPN→T-SDF case. Namely, the KPN/BKPN part can be a finite process that finishes its execution after having consumed a finite number of tokens. It can also present a deadlock or partial deadlock preventing the consumption of tokens. For the case of a timed KPN/BKPN part, the consuming task can also present a throughput slower than the production throughput of the T-SDF part.

Therefore, in a T-SDF→(timed or untimed) KPN/BKPN connection, the write access semantic of the converter

channel allocates a new token in the internal buffer whenever there is room for it. If the internal buffer is full, the following options are available:

(i) *error:* an error is raised and the simulation is stopped;

(ii) *discardOldest:* the next token to be read by the consumer (at the "beginning" of the buffer) is removed, the buffer content is shifted forward and the current token (passed as parameter of the write access) is added to the "end" of the buffer;

(iii) *discardCurrent:* the current token is discarded and the internal buffer remains untouched.

The error option is again set by default. Figure 9 shows an example of a T-SDF→KPN/BKPN conversion, where the T-SDF writing side faces a full internal FIFO at $t = 25$ ms. This example also demonstrates, how the T-SDF side injects timing into the KPN/BKPN part: After unblocking at $t = 27$ ms, the KPN/BKPN part consumes all the token in the internal FIFO. After that, it is blocked until the next writing access of the T-SDF part.

6. APPLICATION EXAMPLE

In this section, we give an example on how converter channels can be used with respect to mixed level simulation and design space exploration. The example deals with the evaluation of a software-defined radio (SDR). An overview of the system is shown in Figure 10, and is basically a simplification of the example given in the introduction. The RF input signal is mixed with a sine wave which has the same frequency as the carrier signal and the result is processed by a lowpass filter. After that, the demodulation is done by software and an analogue demodulator, to compare the results. The input of the software demodulator is an integer with fixed bit width. Algorithm 2 shows the corresponding top-level SystemC code using two converter channels.

Regarding design space exploration and mixed level simulation, this example gives rise to the following tasks (the numbers refer to those in Figure 10):

(1) realising the lowpass filter either as a (behavioral) T-SDF-module or as an electrical network;

(2) realising the software demodulator either as a process network or as a DE module;

(3) varying the bit width of the input of the software demodulator;

(4) realising the analogue demodulator either as a (behavioral) T-SDF-module or as an electrical network.

To make the matter more complicated, any subset of these tasks can be done in parallel. It is obvious that the effort for manual inserting (and adapting) the appropriate converters would be significant. Let us assume an initial model with the lowpass filter and the analogue demodulator modelled as T-SDF modules and the software demodulator modelled within the BKPN MoC, taking `sc_int < bitwidth >` inputs. We then would need a T-SDF→BKPN converter from the output of the lowpass filter to the software modulator, which would also convert double values to `sc_int < bitwidth >` values. Now, executing the tasks above would cause the need for the following respective manual conversion activities.

(1) Realising the lowpass filter as an electrical network: Insert a T-SDF→CT-NET converter between the mixer and the lowpass filter. Replace the initial T-SDF→BKPN converter by a CT-NET→BKPN converter, which also converts `double` to `sc_int < bitwidth >`. Instantiate appropriate signals to connect them.

(2) Realising the software demodulator as a DE module: Replace the initial T-SDF→BKPN converter by a T-SDF→DE converter, which also converts `double` to `sc_int < bitwidth >`. Note that using a SystemC-AMS converter port makes no sense here, since the analogue demodulator is still in the T-SDF domain. Instantiate appropriate signals to connect them.

(3) Varying the bit width of the input of the software demodulator: Change the output of the T-SDF→BKPN converter and the type of the signal which connects it to the software demodulator to `sc_int < new_bitwidth >`. The data type conversion algorithm of the converter must also be altered slightly. Of course, using a parametrisable converter regarding the bit width would make things easier here. In that case this design space exploration task could be steered similar to Algorithm 2.

(4) Realising the analogue demodulator as an electrical network: Insert a T-SDF→CT-NET converter between the lowpass filter and the analogue demodulator. Instantiate an appropriate signal to connect them.

By using converter channels, however, the code for each of the possible variants would be very similar to Algorithm 2. The value of the variable `bitwidth` would change, and the class names of the respective modules (e.g., changing `lowpass_behavioural` to `lowpass_electrical`). This very simple example shows the convenience converter channels offer to the designer.

7. CONCLUSION AND FUTURE WORK

In this article, an overview on the ongoing work on converter channels has been given. Converter channels will provide the designer with a convenient tool to connect system parts using different MoCs and data types. They will quickly and safely solve the syntactical and semantical adaptations that are required by mixed-level simulation and design space exploration. Specifically, the syntactical solution of converter channels will save the manual refinement of those connections. In addition, converter channels provide suitable conversion semantics, which are not always straightforward, since they deal with adaptations in the time, communication, and data type domains. Thus, manual coding for adaptation

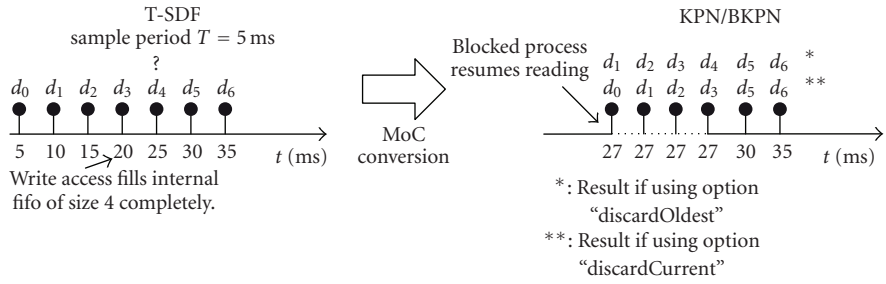

Figure 9: Time-domain behaviour for the T-SDF→KPN and T-SDF→BKPN conversion.

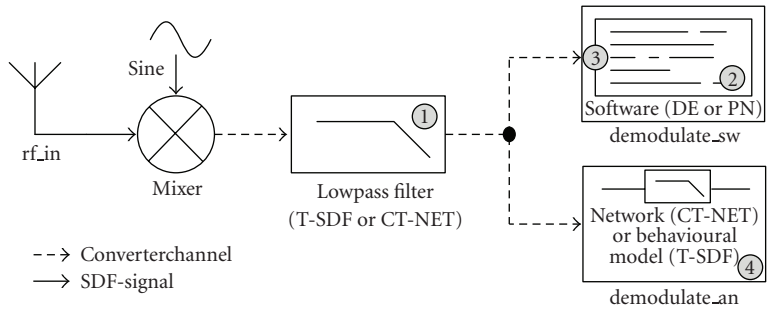

Figure 10: SDR application example.

```
bitwidth = 8;

sca_sdf_signal < double > rf_in; // incoming RF-signal
sca_sdf_signal < double > sine; // sine waverf_in
converterchannel < double > mixedsig;
  // RF-signal multiplied with sine-wave
converterchannel < double, sc_int < bitwidth >> lp_out;
  //output of lp-filter

lp_out.setRangeScaling (0., 1.);
  // assuming the value range within [−1, 1]

mixer mix( "mix"); // multiplies the two input signals
  mix.in1 (rf_in);
  mix.in2 (sine);
  mix.out (mixedsig);

lowpass_behavioural lp ("lp"); // lowpass filter, either
T-SDF-module
  lp.in (mixedsig); // or electrical network
  lp.out (lp_out);

demodulate_sw dem_sw ("dem_sw", bitwidth);
  rec_sw.in (lp_out);
  // software demodulator with sc_int < bitwidth > input

demodulate_an dem_an ("dem_an");
  rec_sw.in (lp_out);
  // analogue demodulator
```

Algorithm 2: SW-defined radio with converter channels.

would be quite time consuming for the system designer, who is usually more interested in the semantic of each system part, than in the glue semantic. The adaptations have been illustrated through several details on the usage of a converter channel and its internal structure, as well as, through a detailed explanation of the semantical issues of the (untimed and timed) KPN/BKPN↔T-SDF connection.

Future work will include finalising the support for further MoC connections, for example, the connection to linear electrical networks (CT-NET). Regarding the connection of CT-NET to T-SDF and DE modules, this is a straightforward task since SystemC-AMS provides many converter facilities for this. Another focus will be the use of converter channels to connect T-SDF models to TLM models.

Despite of ANDRES focussing on run-time reconfigurable systems, converter channels are not meant to adapt to changing communicating facilities during simulation time, for example, to react to a port which changes its bit width. This would go beyond the capabilities of SystemC itself in its current state. Nevertheless, this is an aspect which could be of interest in the future, since this also would allow for an automatic support of a dynamic brand of mixed-level simulation. In this case, the abstraction levels of certain system parts could even be changed during simulation time if the level of detail needed changes in certain situations.

ACKNOWLEDGMENT

This work is supported by the FP6-2005-IST-5 European project.

REFERENCES

[1] Open SystemC Initiative (OSCI), http://www.systemc.org.

[2] A. Vachoux, C. Grimm, and K. Einwich, "Towards analog and mixed-signal SOC design with systemC-AMS," in *Proceedings of the 2nd IEEE International Workshop on Electronic Design, Test and Applications (DELTA '04)*, pp. 97–102, IEEE Computer Society, Perth, Australia, January 2004.

[3] F. Herrera and E. Villar, "A framework for heterogeneous specification and design of electronic embedded systems in SystemC," *ACM Transactions on Design Automation of Electronic Systems*, vol. 12, no. 3, article 22, pp. 1–31, 2007.

[4] L. Cai and D. Gajski, "Transaction level modeling in system level design," Tech. Rep. 03-10, Center for Embedded Computer Systems, University of California, Irvine, Calif, USA, 2003.

[5] C. Grimm, "An introduction to modeling embedded analog/mixed-signal systems using SystemC ams extensions," June 2008.

[6] T. Grötker, S. Liao, G. Martin, and S. Swan, *System Design with SystemC*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[7] C. Grimm, R. Schroll, K. Waldschmidt, and F. Brame, "Mixed-level-simulation heterogener systeme," in *Proceedings of the ITG/GI/GMM-Workshop: Multi-Nature Systems*, Erfurt, Germany, February 2007.

[8] 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*, vol. 17, no. 12, pp. 1217–1229, 1998.

[9] A. Jantsch, *Modeling Embedded Systems and SoC's*, Morgan Kaufmann, San Francisco, Calif, USA, 2004.

[10] I. Sander, "The ForSyDe standard library," Tech. Rep., Royal Technical School, KTH, Kista, Stockholm, April 2003.

[11] Accellera, "Verilog-AMS: Language Reference Manual," version 2.2, November 2004.

[12] P. Frey and D. O'Riordan, "Verilog-AMS: mixed-signal simulation and cross domain connect modules," in *Proceedings of the IEEE/ACM International Workshop on Behavioral Modeling and Simulation (BMAS '00)*, pp. 103–108, Orlando, Fla, USA, October 2000.

[13] IEEE, "IEEE Standard 1076.1: VHDL-AMS Language Reference Manual," 1999.

[14] L. Geppert, "Electronic design automation," *IEEE Spectrum*, vol. 37, no. 1, pp. 70–74, 2000.

[15] C. Brooks, E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, and H. Zheng, "Heterogeneous concurrent modeling and design in java (volume 1: Introduction to ptolemy II)," Tech. Rep. UCB/EECS-2007-7, Department of Electrical Engineering and Computer Science, University of California, Berkeley, Calif, USA, January 2007.

[16] T. Weilkiens, *Systems Engineering with SysML/UML*, Morgan Kaufmann, San Francisco, Calif, USA, 2008.

[17] OMG, OMG SysML Specification, March 2007.

[18] T. A. Jhonson, C. J. J. Paredis, J. M. Jobe, and R. Burkhart, "Modeling continuous system dynamics in SysML," in *Proceedings of the ASME International Mechanical Engineering Congress and Exposition (IMECE '07)*, Seattle, Wash, USA, November 2007.

[19] A. Davare, D. Densmore, T. Meyerowitz, et al., "A next-generation design framework for platform-based design," in *Proceedings of the Conference on Using Hardware Design and Verification Languages (DVCon '07)*, San Jose, Calif, USA, February 2007.

[20] Open SystemC Initiative, SystemC™, http://www.systemc.org.

[21] H. D. Patel and S. K. Shukla, *SystemC Kernel Extensions for Heterogeneous System Modeling*, Springer, New York, NY, USA, 2004.

[22] H. D. Patel, D. Mathaikutty, and S. K. Shukla, "Implementing multi-moc extensions for SystemC: adding CSP and FSM kernels for heterogeneous modelling," Tech. Rep., FERMAT Research Laboratory, Virginia Tech, Blacksburg, Va, USA, June 2004.

[23] H. Al-Junaid and T. Kazmierski, "An Analogue and Mixed-Signal Extension to SystemC," http://eprints.ecs.soton.ac.uk/10644.

[24] S. Orcioni, G. Biagetti, and M. Conti, "SystemC-WMS: mixed signal simulation based on wave exchanges," in *Applications of Specification and Design Languages for SoCs*, pp. 171–185, Springer, New York, NY, USA, 2006.

[25] A. Herrholz, F. Oppenheimer, P. A. Hartmann, et al., "ANDRES-analysis and design of run-time reconfigurable, heterogeneous systems," in *Proceedings of the Design, Automation and Test in Europe (DATE '07)*, pp. 64–71, Nice, France, April 2007.

[26] A. Schallenberg, F. Oppenheimer, and W. Nebel, "Designing for dynamic partially reconfigurable FPGAs with SystemC and OSSS," in *Advances in Design and Specification Languages for SoCs: Part III*, The Chip Design Languages (ChDL) Series, pp. 183–198, Springer, Dordrecht, The Netherlands, 2005, C/C++-Based System Design.

[27] H. Posadas, F. Herrera, V. Fernández, P. Sánchez, E. Villar, and F. Blasco, "Single source design environment for embedded

systems based on SystemC," *Design Automation for Embedded Systems*, vol. 9, no. 4, pp. 293–312, 2004.

[28] N. Savoiu, S. K. Shukla, and R. K. Gupta, "Efficient simulation of synthesis-oriented system level designs," in *Proceedings of the 15th International Symposium on Systems Synthesis (ISSS '02)*, pp. 168–173, Kyoto, Japan, October 2002.

[29] F. Herrera, E. Villar, C. Grimm, M. Damm, and J. Haase, "A general approach to the interoperability of HetSC and SystemC-AMS," in *Proceedings of the Forum on Design Languages (FDL '07)*, Barcelona, Spain, September 2007.

[30] R. Schroll, *Design komplexer heterogener Systeme mit Polymorphen Signalen*, Ph.D. thesis, Institut für Informatik, Universität Frankfurt, Frankfurt, Germany, 2007.

[31] M. Damm, F. Herrera, J. Haase, E. Villar, and C. Grimm, "Using converter channels within a top-down design flow in SystemC," in *Proceedings of the 15th Austrian Workshop on Microelectronics (Austrochip '07)*, Graz, Austria, October 2007.

[32] F. Herrera, P. Sanchez, and E. Villar, "Heterogeneous system-level specification in SystemC," in *Advances in Design and Specification Languages for SoCs*, P. Boulet, Ed., CHDL Series, Springer, New York, NY, USA, 2005.

[33] F. Herrera and E. Villar, "Mixing synchronous reactive and untimed mocs in SystemC," in *Applications of Specification and Design Languages for SoCs*, A. Vachoux, Ed., CHDL Series, Springer, New York, NY, USA, 2006.