A Framework for Embedded System Specification under Different Models of Computation in SystemC

F.Herrera
University of Cantabria
ETSIIT, TEISA Dpt. 39005
Santander, Spain
+34 942 20 08 78
fherrera@teisa.unican.es

E.Villar
University of Cantabria
ETSIIT, TEISA Dpt. 39005
Santander, Spain
+34 942 20 13 98
villar@teisa.unican.es

ABSTRACT
This paper presents a heterogeneous specification methodology built on top of the standard SystemC kernel. The methodology enables abstract specification supporting heterogeneity, which in this context entails the ability to describe and connect parts of the system specification under different models of computation (MoCs). A main and distinguishing contribution of the methodology is that the support is provided while maintaining the standard kernel of SystemC unchanged, by means of a set of specification rules and a heterogeneous support library built on top of the SystemC standard library. This is possible thanks to an abstraction technique that can integrate any new MoC that can be abstracted over the underlying discrete-event simulation kernel. Primitives, guidelines and rules of the specification methodology, including those related to heterogeneous support, and the basis of the abstraction technique are described. Experimental results demonstrate the benefits of the methodology.

Categories and Subject Descriptors
F.1.1 [**Models of Computation**]: Relations between models.

F.3.1 **[Theory of Computation]**: Specifying and Verifying and Reasoning about Programs - *Specification Techniques*.

I.6.5 [**Simulation and Modeling**]: Model Development - *Modeling Methodologies*.

General Terms
Design, Languages.

Keywords
Heterogeneous Specification, SystemC.

1. INTRODUCTION
Due to the increasing integration capability, nowadays, embedded systems are able to integrate a greater diversity of components (specific hardware blocks, DSPs, general purpose processors, etc). This makes heterogeneous specification much more important in any design methodology for this kind of systems. At the system specification level, heterogeneity is conceived as the ability to build a specification as a set of communicating subsystems described under different models of computation (MoCs) [1].

DAC 2006, July 24–28, 2006, San Francisco, California, USA.

The ability to select a particular MoC for the specification of a system part or subsystem, facilitates faster development of the specification, since the primitives provided by the MoC adapt to the abstraction level required in each main aspect of the specification. This is named fidelity in [2]. In [3], four aspects called *domains: Data*, *Computation*, *Communication* and *Time* enable the analysis and classification of different MoCs. In addition, the ability to use MoCs with higher abstraction in one or more domains improves simulation speed [3]. This is especially noticeable when abstraction is carried out in the *Time* domain, although abstraction in the Data domain can provide remarkable speed-ups too. A further benefit is that the specification under the restrictions of a particular MoC can guarantee some specific properties, such as determinism, protection against deadlock, etc, which can be advantageous in many applications. Finally, the support of heterogeneity can facilitate implementation flow for heterogeneous platforms since the specification primitives of each subsystem are syntactically and/or semantically closer to the platform components. Thus, software generation or hardware synthesis becomes closer to a mapping procedure.

In this paper a heterogeneous specification methodology built on top of standard SystemC is presented. The methodology provides all of the previous advantages and, as will be seen, improves the support provided by other methodologies in several aspects. The structure of the paper is as follows. First, in section 2, the related work is reviewed. In section 3, the specification methodology is explained. Section 4 presents experimental results with a real and significant example. Section 5 ends with the main conclusions.

2. RELATED WORK
An important framework which provides a graphical way to develop a heterogeneous specification is Ptolemy II [4], which takes Java as the implementation language. In Ptolemy II, each component bounds a MoC, defined by a *director* class (MoC-component match) and each MoC runs over its own specific simulation engine. Hierarchical design is possible depending on the MoCs combined in the hierarchical structure. A major drawback is speed, since these models run over a virtual machine. In addition, Ptolemy II is a modeling and simulation framework, although there are some works that tackle the implementation flow [5]. In [6], a methodology that supports heterogeneous design and that proposes an implementation flow through refinement is shown. However, this methodology is based on

Haskell, a functional language whose use is not widespread among embedded system designers. Similarly, Metropolis [7] is an integrated design framework which uses a metamodel language for specification. This framework is able to support not only specification and simulation, but also other design activities, such as, formal analysis and implementation. It also differs from Ptolemy in which heterogeneity is supported by the different semantics of the communication primitives (called *media*), instead of by providing MoC specific simulation kernels.

SystemC [8][9], a C++ library, is currently the most widely accepted proposal as a unified system-level specification language. Many developers are already familiar with C language and SystemC provides good simulation efficiency. From its beginnings, SystemC has proven to be suitable for HW description at RT or Behavioral level [10]. In addition, since its 2.0 release, some of its standard primitives provide a basic, but still insufficient, support for modeling under more abstract MoCs (i.e. for PN MoC [11]). However, an important advantage of SystemC is that it provides features (such as *notify* and *wait* primitives, or the *sc_interface*, *sc_channel* and *sc_prim_channel* classes) which enable the extension of the language to cleanly and efficiently provide customizable MoCs over the "base" MoC of the SystemC kernel [8]. Thus, new specification primitives (i.e. channels) and features can be provided and grouped in a methodological-specific library for heterogeneous support. This would keep the SystemC kernel small, general and standard. However, this point was not developed in [8]. The standard TLM library [12] proposes a reduced set of general and simple interfaces and some communication elements. The most important part, as stated in [12], is the set of interfaces. TLM interfaces provide TLM prototypes with homogeneity, and so reusability, to speed up architecture exploration. However, they do not define the complete semantics of communication mechanisms and, therefore, the semantics of the system. Moreover, a system-level specification methodology requires a set of specification guidelines and rules, elements to oblige or check their accomplishment and a set of MoC primitives with their particular, complete and unambiguous semantic content (which, for instance, can be executed by the simulation kernel). Thus, SystemC and its related standard libraries still lack a complete and methodic support for heterogeneous specification.

In [2], the SystemC kernel extension for the support of heterogeneity is proposed. Kernel extension means that a set of new C++ classes enhances the set already provided by the SystemC kernel. Some of them are public and provide the specifier with new MoC specific primitives. However, other classes are transparent for the specifier and extend the simulation kernel to provide, as in Ptolemy, new simulation engines, each one adapted to its corresponding MoC. The main drawback of kernel extension techniques is that the new C++ classes are provided in a way which obliges the inclusion of either a larger non-standard SystemC kernel or several new ones. That is, it does not take full advantage of SystemC features for MoC extension. It also has to be considered that simulation speed-ups reported by these techniques are limited, as with any other technique for heterogeneity support, by the level of detail required for each domain in each subsystem, and thus by Amdahl's law. SystemC-AMS [13] covers the specification under continuous-time MoCs, which seem to more clearly justify the extension of the kernel.

In this paper, the work developed in [14][15][16], towards a heterogeneous specification methodology in SystemC is completed and summarized. The methodology is able to support the specification under several MoCs whose restrictions, primitives and properties provided are well-known. Among these MoCs, there are untimed MoCs, such as Process Networks (PN) [11], Kahn Process Networks (KPN) [17], Communicating Sequential Processes (CSP) [18], Synchronous Data flow (SDF) [19] MoCs or Synchronous MoCs, such as the Synchronous Reactive (SR) [20] MoC. Timed MoCs are also supported, considering among them, the refinements performed from previous ones, i.e., from a fifo channel. An important point of this methodology is that, as Metropolis, it tries to provide an integrated design methodology. That is, as well as specification and simulation, other design activities, such as, system-level performance estimation through simulation and automatic SW implementation are supported [21]. The proposed methodology is open, it can easily integrate any new MoC that can be simulated in C++ and can cooperate with a kernel extension technique.

3. SPECIFICATION METHODOLOGY

The specification methodology presents a first level where the specification primitives are contained in the current kernel library. A set of rules and guidelines make feasible the writing of an executable specification as shown in Figure 1, which is ready for the application of the methodology of [21].

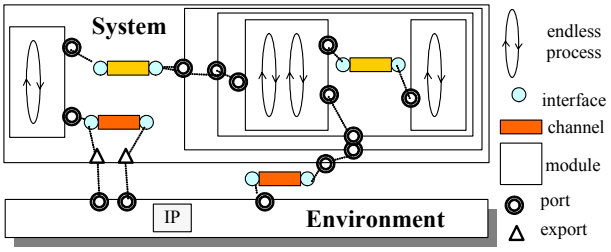

Figure 1. Primitives of the specification.

The structure of the specification is static (generated before the start of simulation) and exhibits at least one hierarchical level, although more levels are possible. The top-level is instantiated from a *sc_main* function, which encloses the test bench and the system modules (*sc_module* primitive). Environment and system are communicated by means of channels, which become the input/output of the system. Since the 2.1 version, the *sc_export* primitive has enabled the inclusion of the I/O channels within the system module. Regarding the basic rules and methodological guidelines, the only restriction on the environment module(s), besides those imposed by SystemC, is the need to have ports compatible with the channels which connect system and environment modules (I/O Channels). Restrictions on the system specification focus the task of the designer on writing concurrent functionality through C/C++ algorithms (thus, on writing computation code). Concurrency is supported since the functionality is contained by SystemC processes (*SC_THREAD* or *SC_METHOD*). There is a strict separation between communication and computation since channels are the only way for processes to communicate. The methodology makes it possible to specify communication by means of standard or library provided channels. The specifier only needs to know channel syntax and semantics, to instantiate and access them. Explicit synchronization through wait (except wait on time delay) and notify primitives is not allowed, since these cannot be used in the computation code. The use of new user channels at system level is possible, but associated implementations have to be

provided for the support of system-level profiling and SW/HW generation steps [21]. Communication among processes of different module instantiations is through ports (*sc_port* or *sc_export*). IPs of any class can also be included as modules of the environment code. The IP must have compatible ports otherwise the development of a wrapper is necessary. Additional information can be included for other design activities, such as debugging, performance estimation or SW synthesis (i.e., *#define* preprocessor clauses are used to specify the HW/SW partition).

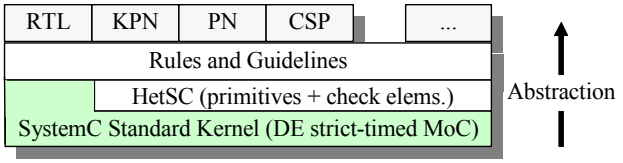

Figure 2. Heterogeneity is preserving the standard kernel.

Heterogeneity is supported through an additional level of rules and primitives, whenever the standard SystemC support is not sufficient (called lack of expressiveness in [2]). Those elements and primitives are grouped in a library called HetSC. The public SystemC primitives provided by the HetSC library are macros, SystemC interfaces and mostly SystemC channels providing MoC-specific communication semantics, with semantics and/or checking features not provided by the SystemC standard channels. Check elements monitor the fulfillment of MoC constraints and specific MoC primitives. For the user, they are usually transparent classes, but there is also additional code embedded in the implementation of HetSC channels. In contrast to the TLM library, the most important content of the HetSC library is the set of communication channels with all their specific semantic content. Indeed, HetSC is mainly a communication library which could be made compatible with TLM interfaces. Specification guidelines and rules complement the specification methodology especially in those points where checking features provided cannot oblige the specifier to fulfill MoC restrictions. MoC condition checkers can be enabled and disabled to allow some flexibility in the application of MoC rules.

Table 1. Some elements provided by HetSC.

Public	Macros	*CH_MONITOR*, *SDF_NODE*,…
	Interfaces	*uc_rv_if, uc_rv_sync_if*,…
	Channels	*uc_fifo*, *uc_inf_fifo*, *uc_arc*, *uc_rv*,…
	BCs	*uc_fifo_inf_SR*, *uc_inf_fifo_signal*, …
Hidden	Check class	*system_memory_monitor*, …

Most untimed MoCs [3] supported (KPN/PN/CSP/SDF) are characterized by a homogeneous computation style, that is, a sequential algorithm, which is mapped to the SC_THREAD. Untimed MoCs are mainly distinguished amongst themselves by their communication domain. The PN and KPN MoCs are characterized by the fifo channel. SystemC can provide a basic coverage for these MoCs through the *sc_fifo* channel. However, their support has been improved through two new channels provided by the HetSC library, the *uc_fifo* (for PN) and the *uc_inf_fifo* (for KPN). These new channels implement checks, which monitor whether several writers or readers access a channel instantiation or ban internal data introspection. Other HetSC primitive channels cover the lack of SystemC primitives matching the semantics of constructors belonging to other MoCs, as is the case of CSP and SDF MoCs. For CSP, a set of three rendezvous channels (*uc_rv_sync*, *uc_rv_uni* and *uc_rv*) is provided. In the SDF case, the *uc_arc* channel also serves to acquire the additional information about the specification which the SDF MoC requires: the production and consumption rates of the SDF graph arcs [19]. At execution time, the *uc_arc* uses the rate figures to condition the execution semantics, i.e. the consumption rate fixes when to unblock the process where the SDF node is mapped. Support for the SR MoC imposes more restrictive conditions on the event ordering. Specifically, the SR MoC imposes the perfect synchrony hypothesis [20]. That is, the system reacts instantaneously and outputs are synchronous (they have the same time tags) with the inputs which provoked them. As a consequence, all the activity of the system concentrates on specific points of the time axis called slots [3]. In the methodology, a new *uc_SR* channel provides a way to connect processes in what is called a *reactive chain*, where every event has the same SystemC simulation time for the same reaction. Two new important checks are provided. One ensures a time separation between the write accesses which originate the reaction. The other prevents the reaction involving a time advance. The support for clocked and timed MoCs provided by the standard kernel and AMS extension is completed through some channels. These enable a direct refinement toward signal-based or clocked signal-based protocols. For instance, the *uc_sigclocked_fifo* is a refined version of a *sc_fifo* channel with signal accesses synchronized through a channel's clock signal.

As subsystems under different MoCs usually communicate among themselves, heterogeneous specification requires the definition of MoC interfaces. In the methodology, MoC interfaces are located in border channels (BCs) and/or processes (BPs). These are SystemC primitives which share (with the rest of specification primitives) the underlying simulation kernel. Thus no element outside SystemC has to be used to perform the MoC adaptation. A Border Process (BP) is a process accessing channels belonging to different MoC parts, where the adaptation is explicitly written in the body of the function associated to the BP. The BP is a highly flexible mechanism which can be used by the specifier to mix two or more MoCs. A Border Channel (BC) is a SystemC channel where the MoC interface is concentrated. The adaptation is hidden and performed within channel implementation. The basic principle of the BC semantic is that from each MoC side, the BC is seen as the channel associated to that MoC.

4. EXPERIMENTAL RESULTS

A SystemC EFR Vocoder fulfilling the GSM 06.60 standard [22] has been developed. This voice codec maps input blocks of 160 13-bit speech samples to encoded blocks of 244 bits and vice versa in the reverse sense (synthesis). The standard provides an ANSI-C reference code of 15,000 code lines and a set of test benches for validation through simulation [22]. The SystemC specification was developed as an adaptation of the original ANSI-C code, providing it with hierarchy and concurrency. 83.3% of code lines of the reference ANSI-C code (12,500 lines) were reused. The specification has a system and an environment module. The system module is composed of two main modules, the EFRcoder and the EFRdecoder. Both, coder and decoder hide a new hierarchy level. The system comprises a total of 14 modules and 13 processes. These processes are SC_THREADs communicated by infinite fifo channels (*uc_inf_fifo*) constituting

a Kahn process network. This KPN version avoids an important number of blocking situations and deadlock conditions which could appear in a PN version of unknown timing. For the 28 tests, no simulation time of the SystemC specification exceeded that of the ANSI-C model execution time by more than 9%.

After the single-MoC approach, experiments to refine the example and to mix several MoCs in the same specification were done. The KPN approach provided two significant figures for the refinement, the maximum fifo sizes (considering a timed environment providing a voice sample every 0.125μs) and the maximum memory size if a shared channel memory were employed instead. Maximum fifo sizes were employed to develop a whole PN version, substituting every *uc_inf_fifo* channel instantiation by a *uc_fifo* channel instantiation. The PN specification also ran without deadlocks and fulfilling the tests. An advantage of a correct PN specification is that, even when vocoder output timing is not accomplished (the vocoder being too slow), the functional tests still pass. The maximum shared memory size was used to generate an alternative specification with a shared channel memory for the coder, thus having a mixed KPN(coder)–PN (decoder) specification. After that, two modules of the output part of the coder were refined to a Timed Clocked PN style, making them ready for a HW implementation flow. For the connection to the Timed Clocked MoC, *uc_inf_fifo_signal* and *uc_signal_inf_fifo* border channels where instantiated at a different hierarchical level. The communication between two Timed Clocked modules was done through a *sc_sigclocked_channel*, which provides a refined Timed Clocked version of the previous fifo instantiation. The last experiment was the substitution of several fifo channel instantiations inside the coder specification, used for synchronizing several modules with the in-band reset condition, by rendezvous channel instantiations. That substitution provided a more natural specification since those modules have to check if they have to be reset once another module has processed the last input frame to detect if it was an in-band reset frame (homing frame). The inner processes accessing both, *uc_inf_fifo* and rendezvous channels were BPs, while the *uc_inf_fifo_signal* and *uc_signal_inf_fifo* channels were BCs.

5. CONCLUSIONS

This paper reports a framework for system-level heterogeneous specification, built over the small, compact and standard SystemC kernel. The methodology is able to provide support for any MoC that can be abstracted from the underlying DE simulation kernel and focuses on MoCs whose restrictions and properties are stated. Each MoC is supported by rules, guidelines and, if necessary, new primitives, which are packed in a new methodological library, HetSC. Finally, although the methodology has made a contribution to the understanding of the temporal semantics of the system, future work will establish the formal metamodel and semantics of the specification.

6. ACKNOWLEDGEMENTS

Work supported by the Spanish MEC (TEC 2005-03301 project).

7. REFERENCES

[1] E. Lee, A. Sangiovanni-Vincentelli. “A Framework for comparing Models of Computation”. IEEE Trans. on CAD of ICs and Systems, V.17, N.12, December 1998.

[2] H.D.Patel, S.K.Shukla. “SystemC kernel extensions for Heterogeneous System Modelling: A framework for Multi-MoC Modelling and Simulation”. Kluwer. Aug. 2004.

[3] Axel Jantsch. “Modeling Embedded Systems and SoC's”. Morgan Kaufman Publishers. Elsevier Science. 2004.

[4] PtolemyII. http://ptolemy.eecs.berkeley.edu/ptolemyII

[5] J.L.Pino, S.Ha, E.A.Lee, J.T.Buck. “SW Synthesis for DSP using Ptolemy”. Journal on VLSI Processing, vol. 9, no.1, pp 7-21. January. 1995.

[6] I.Sanders. “System Modeling and Design Refinement in ForSyDe”. Thesis. KTH Univresity. Stockholm 2003.

[7] F.Balarin et al. “Metropolis: An Integrated Electronic System Design Environment”. IEEE Computer Magaz. 2003.

[8] T. Grötker, S. Liao, G. Martín, S. Swan. “System Design with SystemC”. Kluwer; 2002.

[9] “SystemC: Methodologies and Applications”. Ed. W.Mueller, W. Rosenstiel, J.Ruf. Kluwer. March 2003.

[10] Synopsys Inc. “Describing Synthesizable RTL in SystemC”. Version 1.2. November 2002.

[11] E.A.Lee, T.M.Park. “Dataflow Process Networks”. Proceedings of the IEEE, 1995.

[12] A.Rose, S.Swan, J.Pierce, J.M.Fernández. “Transaction Level Modeling in SystemC”. Av. at www.systemc.org.

[13] Website. www.systemc-ams.org

[14] F.Herrera, P.Sánchez, E,Villar. ”Modeling and Design of CSP, KPN and SR Systems with SystemC”, in C.Grimm (Ed.) “Languages for System Specification”. Kluwer, 2005.

[15] F.Herrera, P.Sánchez, E,Villar. ”Heterogeneous System-Level Specification in SystemC” in P.Boulet (Ed.) “Advances in Design and Specification Languages for SoCs”. Kluwer, 2005. See www.teisa.unican.es/HetSC.

[16] F.Herrera, E.Villar. “Mixing Synchronous Reactive and Untimed Models of Computation in SystemC”. In proc. Of Forum of Design Languages. FDL’05. Sept, 2005.

[17] G. Kahn. ”The Semantics of a simple Language for Parallel Programming”. Proc. IFIP 74, North-Holland, 1974.

[18] C.A.R. Hoare. “Communicating Sequential Processes”. Communications of the ACM, V.21, N.8, August 1978.

[19] E.A.Lee, D.G. Messerschmitt “Synchronous Data Flow”. Proc. of the IEEE, Vol75, No.9,September, 1987.

[20] EA. Benveniste & G. Berry. “The Synchronous Approach to Reactive and Real-Time Systems”. Proceedings of the IEEE, V.79, N.9, September, 1991.

[21] H.Posadas, F.Herrera, V.Fernández, P.Sánchez & E.Villar. “Single Source Design Environment for Embedded Systems based on SystemC”. Design Automation for Embedded Systems Journal. Springer. Dec. 2004. V.9, N.4. pp.293-312.

[22] ETSI/EN 301 245, ETSI/EN 301 244 and ETSI/EN 301 250 standards. 1998. Available at www.etsi.com.