HetMoC: Heterogeneous Modelling in SystemC

Jun Zhu, Ingo Sander, Axel Jantsch
Royal Institute of Technology, Stockholm, Sweden
{junz, ingo, axel}@kth.se

***Abstract*—We propose a novel heterogeneous model-of-computation (HetMoC) framework in SystemC for embedded computing systems. As the main contribution, we formally define the computation and communication in multiple domains (continuous-time, discrete-event, synchronous/reactive, and untimed) as polymorphic *processes* and *signals*, and present domain interfaces to integrate different domains together for heterogeneous process networks. Especially, the continuous-time signals are defined with time *continuum*, which are distinguished from existing approaches. For implementation, a functional modelling style has been adopted to construct HetMoC. A solver with error estimation has been exploited in numerical approximation, and the time-varying functionalities in adaptive systems have been captured in HetMoC as well. In experiments, based on an adaptive transceiver system case study, HetMoC shows promising capabilities compared with a reference model in SystemC-AMS.**

I. Introduction

The heterogeneity and rapid time-to-market demand of consumer electronics are expected to challenge the system-level modelling and design methodologies for embedded computing systems, and have motivated researchers to pursue new design paradigms to bridge the industry productivity gap between design complexity and design capacity. To support co-specification and co-design in a mixed analogue and digital circumstance, system-level design approaches are required to capture heterogeneous embedded systems in a common framework [1, 2].

As a pioneer work, Ptolemy II [3] is such a modelling environment. Based on the abstract semantics of computation modules (called *actors*), a prototype has been proposed to combine the heterogeneous modelling styles in continuous-time (CT), discrete-event (DE), and synchronous/reactive (SR) models-of-computation (MoCs) [4]. While Ptolemy II is implemented in Java and fits concurrent software design, hardware platform modelling is essentially not covered [1]. It is argued that the potential solution to system-level specifications involves a heterogeneous mixture of hardware/software, and requires in addition extensive supports on design refinement, platform characterization, and hardware synthesis. From the implementation perspective, SystemC based approaches are very prominent for their inherent support on concurrent hardware/software design and orthogonality (strict separation) between computation and communication. However, to specify heterogeneous systems based on the DE simulation kernel of SystemC is not straightforward.

Related work. Several libraries have been proposed as extensions to the SystemC standard library to support heterogeneous specifications. HetSC [5] provides a set of rules and guidelines to integrate different MoCs on top of the SystemC DE simulation kernel, in which the *temporal* causality among DE events are broken into *ordered* relationship for untimed specifications. To capture the CT dynamics, Vachoux et al. [6] propose a SystemC-AMS extension. Based on a timed synchronous data flow (T-SDF)[1] *synchronization layer*, different MoCs can be integrated in SystemC-AMS on top of the multi-rate T-SDF simulator. However, the current SystemC-AMS is only a step towards a seamless CT modelling framework. For instance, the CT solvers to be plugged-in are limited to those with fixed sampling time period, caused by the fixed timing intervals in T-SDF models. The numerical stability of the existing solvers and simulation accuracy need to be further investigated. To keep system-level synthesis in mind, OSSS+R [8] is a synthesizable extension to SystemC in clocked-SR domain, which targets dynamically reconfigurable FPGA platforms. To integrate the different SystemC extensions above, Damm et al. [9] have used converter channels for heterogeneous MoCs connection. Although converter channels leverage the simulation capabilities over a wide spectrum of MoCs, it is an *ad-hoc* approach to glue existing legacy modelling styles. There is another approach which extends the SystemC simulation kernel [10], however, so far only DE and untimed models have been integrated. It is hard to integrate this approach with the others since it is based on a modified SystemC kernel.

Contributions. In this paper, we present a formal heterogeneous model-of-computation (HetMoC) framework in SystemC, which complements the existing approaches [5, 6, 10]. A network of heterogeneous processes communicate with each other via signals (implemented as domain-polymorphic FIFO channels), and different domains are integrated using domain interfaces. We propose a novel modelling style for CT MoC with pure CT dynamics, based on which stepwise abstraction is made to build other model domains. Instead, the heterogeneous framework built in Ptolemy II is based on the enhancing of SR semantics [4]. Besides an integration of SR, DE, and CT MoCs inspired by Ptolemy II, HetMoC supports the mixing with untimed models as well. However, here we only discuss the specialized untimed synchronous dataflow (SDF) models, driven by our case study in experiments and the limited space.

The remainder of this paper is structured as follows: we specify the HetMoC framework formally in Section II, and present its implementation in SystemC in Section III. The experiments on an adaptive transceiver system case study within both SystemC-AMS and our HetMoC are in Section IV. Finally, Section V concludes the paper.

[1]Synchronous dataflow (SDF) model [7] is a restricted form of dataflow model, in which the numbers of data tokens produced and consumed by processes during each evaluation are constant.

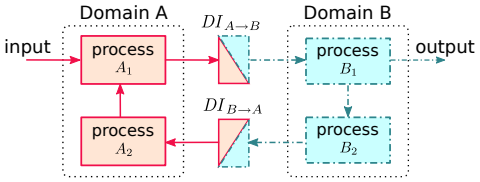
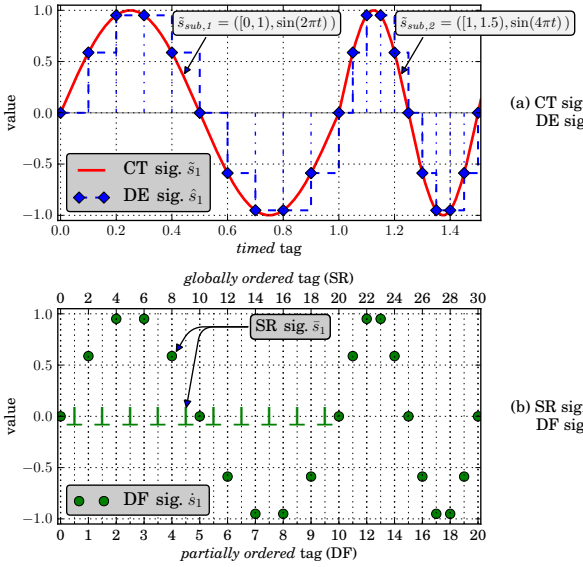


Fig. 1: A heterogeneous process network, in which $DI_{A\to B}$ stands for the domain interface from A to B and vice versa.



(a) CT signal $\tilde{s}_1$ and DE signal $\hat{s}_1$.

(b) SR signal $\bar{s}_1$ and DF signal $\dot{s}_1$.

Fig. 2: Visualization of signals in different MoCs domains. The DE ($\hat{s}_1$), SR ($\bar{s}_1$), and DF ($\dot{s}_1$) signals are based on a stepwise abstraction of the CT signal $\tilde{s}_1$.

II. Modelling Framework

We present our modelling framework HetMoC for specification and simulation of heterogeneous distributed systems. The designer captures a target system as a process network, as illustrated in Fig. 1. In the graph, blocks are *processes* to specify computation and edges are *signals* to connect processes. The *processes* and *signals* are diversified in our framework for heterogeneous MoCs, instead of a single Kahn process network domain as in [11]. Different computation domains interact with each other via domain interfaces (DIs).

A. Methodology Overview

While our framework HetMoC has a sound formal base to integrate heterogeneous modelling domains. it intends to be user-friendly for end users. Based on a clear separation of concerns between computation and communication, the end users only need to specify process *functionality* and (both intra- and inter-domain) *interconnections*, without being aware of the internal formal definitions to support the heterogeneous modelling environment.

Let $\mathbb{N}_0$ denote the set of natural numbers including 0, $\mathbb{R}$ the set of real numbers, and $\mathbb{R}_+$ the set of nonnegative real numbers. The precise definitions of *processes*, *signals*, and *domain interfaces* for HetMoC are formalized in the following.

B. Signals

Signals connect computation processes to construct complex systems, and are the only way processes may communicate. A signal $s : \mathbb{T} \to \mathbb{V}$ is a functional mapping from tag series $\mathbb{T}$ to a value set $\mathbb{V}$ [12], i.e., $s(t)$ is a value of the signal at a specified time tag t. The domain of $\mathbb{V}$ is application-dependant and can be any primitive or composite data-types (e.g., *boolean*, *integer*, and *user-defined data structure*). The domain of $\mathbb{T}$ is MoC-dependant and may consist of real or integer numbers to model the time or precedence properties. Usually, a signal is denoted as a sequence of data events, and may contain less (or even none) timing information upon the increasing abstraction levels in different MoCs.

1) CT signals: A CT signal $\tilde{s}$ is defined over a continuous series of time and is expressed as a concatenation of consecutive sub-signals (CT specific events). In this sense, each event in CT signals is a sub-signal $\tilde{s}_{sub,n}$, which is an individual function $f_n : \mathbb{R}_+ \to \mathbb{V}$ defined over a specified time interval I_n. An interval I_n is a convex subset of $\mathbb{R}_+$, which may be open, half-open, or closed. For instance, $\tilde{s}_{sub,n} = ([a_n, b_n), f_n)$ denotes a sub-signal with f_n defined over a half-open interval $I_n = [a_n, b_n)$, in which $a \in \mathbb{R}_+$ is the left end-point, $b \in \mathbb{R}_+$ the right end-point, and $a < b$. The time duration $|I_n|$, i.e., $b_n - a_n$, can be either finite or infinite, as long as there is no gap or overlap between the time intervals of sequencing sub-signals.In this paper, we assume that all CT signals have the same start time 0 and $\mathbb{V} = \mathbb{R}$ for numerical computation. A pictorial representation of the value $\tilde{s}_1(t)$ at each continuous time tag t for an example CT signal $\tilde{s}_1 = \{\tilde{s}_{sub,1}, \tilde{s}_{sub,2}\}$ is in Fig. 2 (a), in which $\tilde{s}_1$ has varying frequencies (1–2 HZ) in two consecutive *sine* sub-signals.

2) DE signals: A DE signal $\hat{s} : \mathbb{R}_+ \to \mathbb{V}$ is an abstraction (sampling) on a CT signal $\tilde{s}$, and consists of a sequence of DE events, i.e., $\hat{s} = \{\hat{e}_0, \hat{e}_1, \cdots, \hat{e}_n, \cdots\}, \forall n \in \mathbb{N}_0$. Each event $\hat{e}_n = (t_n, v_n)$ is a value v_n defined upon a real number t_n to measure the passage of time. The time elapsed between two consecutive events is not fixed and can be any real number. Both CT and DE signals are *timed*. An example DE signal $\hat{s}_1$ which is a sampling on $\tilde{s}_1$ is illustrated in Fig. 2 (a). Given 10 samples in each period of the CT *sine* waves and $\hat{s}_1(t) = \tilde{s}_1(t)$, the sampling rates vary for different sub-signals $\tilde{s}_{sub,1}$ and $\tilde{s}_{sub,1}$.

3) SR signals: A SR signal $\bar{s} : \mathbb{N}_0 \to \mathbb{V}_\perp$ abstracts away the passage of time from DE signals, i.e., a SR event has no timing tag. Instead only the precedence properties are implicitly preserved by the order of events, and the value domain $\mathbb{V}_\perp = \mathbb{V} \cup \{\perp\}$ is a set of absent $\perp$ extended values to maintain globally ordered synchrony. That is, the signal events are $\perp$ when the values are absent. Although SR signals contain no time information, a fixed or varying time can be associated with the clock ticks between events in interpretation. An SR signal $\bar{s}_1$, which uses a fixed implicit time tick $\frac{1}{10}$ to abstract from $\hat{s}_1$, is exemplified in Fig. 2 (b). Although the time information in $\hat{s}_1$ is lost in $\bar{s}_1$, data events in $\bar{s}_1$ are globally ordered by inserting $\perp$ events for synchronization. Here, we simply assume that events are absent when values are not defined at the implicit time in $\hat{s}_1$. Especially, different SR signals may have multi-clock in heterogeneous systems,

as in Lustre [13], Signal/Polychrony [14], and our previous work [15, 16].

4) DF signals: An untimed dataflow (DF) signal $\dot{s} : \mathbb{N}_0 \to \mathbb{V}$ in functional level keeps only a partial order of events for each signal, which is a sequence of data token values. A typical implementation of DF signals is asynchronous FIFO queues. Based on the abstraction of the SR signal $\bar{s}_1$, a DF signal $\dot{s}_1$ is illustrated in Fig. 2 (b).

C. Domain interfaces

Signals in different MoCs can be distinguished by the carrying time or order information, but they are domain polymorphic and can not communicate with each other directly. To transfer signals across MoC domains, *domain interfaces* are introduced as the glue modules when domain polymorphism exists.

A domain interface $DI_{A\to B}$ is a functional mapping from signals in domain A to B, given the necessary user specified parameters. For instance, given a DE signal $\hat{s}_T$ which specifies the sampling time stamp (with dummy values in events) on a CT signal, a domain interface from CT to DE domain is defined as $DI_{CT\to DE} : \hat{s}_T \times \tilde{s} \to \hat{s}$. Similarly, a domain interface from DE to SR domain is $DI_{DE\to SR} : \bar{s}_T \times \hat{s} \to \bar{s}$, in which the event values of $\bar{s}_T$ specify the implicit time of synchronous clock ticks.

Domain interfaces are composable. That is, a domain interface from CT to SR domain $DI_{CT\to SR} : \hat{s}_T \times \bar{s}_T \times \hat{s} \to \bar{s}$ can be defined as *curried*[2] function as follows.

$$DI_{CT\to SR}(\hat{s}_T, \bar{s}_T) = DI_{DE\to SR}(\bar{s}_T) \circ DI_{CT\to DE}(\hat{s}_T) \tag{1}$$

in which $\circ$ is a function composition operator, with the definition $g \circ f(s_1) = g(f(s_1))$. For different MoCs, domain interfaces preserve the *causality* and *monotonicity* of signals [12].

D. Processes

A process specifies computation, and is a functional mapping from input signals to output signals. After each evaluation, a producer process generates output events which are sent via signals along communication channels to consumer processes. Each signal has one producer and one consumer process.

1) CT processes: To enforce CT computation, a combinatorial process *combCT*(f) simply applies a specified function f in curried form on sub-signals of the input signal. Since sub-signals keep time *continuum*, the computation has a pure CT dynamics, which makes HetMoC distinguished from SystemC-AMS Vachoux et al. [6] and Ptolemy II [4]. Accordingly, a specialized process *scaleCT*(k) to scale input signal by factor k can be defined in a functional style.

$$\begin{aligned} &\textit{scaleCT}(k) = \textit{combCT}(f) \\ &\quad \textit{where } (f(g))(x) = k \cdot g(x), \quad k \in \mathbb{R}, g \in \mathbb{R} \to \mathbb{R} \end{aligned} \tag{2}$$

in which, g is an implicit auxiliary function. For instance, *scaleCT*(2) applied to the signal $\tilde{s}_1$ in Fig. 2 (a) results in

$$\textit{scaleCT}(2)(\tilde{s}_1) = \{([0,1), 2\sin(2\pi t)\,), ([1,1.5), 2\sin(4\pi t)\,)\}$$

Furthermore, the definitions of combinatorial processes with one-input can be generalized to multiple-input multiple-output (MIMO) signals. Without losing generality, we show how a process *comb2CT*(f) that combines two input signals with a binary (2-input) function f can be constructed in the following. Given $p =$ *comb2CT*(f), the definition relies on a recursive partitioning and computation on input signals:

$$p(\tilde{s}_a, \tilde{s}_b) = \{\}, \text{ if either } \tilde{s}_a \text{ or } \tilde{s}_b \text{ is empty} \tag{3a}$$

$$p((f_{a,1}, I_{a,1}) : \tilde{s}'_a, (f_{b,1}, I_{b,1}) : \tilde{s}'_b)$$

$$= \begin{cases} (f(f_{a,1}, f_{b,1}), I_{a,1}) : p(\tilde{s}'_a, \tilde{s}'_b), & \text{if } |I_{a,1}| = |I_{b,1}| \\ (f(f_{a,1}, f_{b,1}), I_{a,1}) : & \\ p(\tilde{s}'_a, (f_{b,1}, I_{b,1} \setminus I_{a,1}) : \tilde{s}'_b), & \text{if } |I_{a,1}| < |I_{b,1}| \\ (f(f_{a,1}, f_{b,1}), I_{b,1}) : & \\ p((f_{a,1}, I_{a,1} \setminus I_{b,1}) : \tilde{s}'_a, \tilde{s}'_b), & \text{if } |I_{a,1}| > |I_{b,1}| \end{cases} \tag{3b}$$

in which the operator : determines that the sub-signal ahead is the head of a signal, i.e., $(f_{a,1}, I_{a,1}) : \{(f_{a,2}, I_{a,2})\} = \{(f_{a,1}, I_{a,1}), (f_{a,2}, I_{a,2})\}$, and $\tilde{s}'_a$ and $\tilde{s}'_b$ are the remainders of signal $\tilde{s}_a$ and $\tilde{s}_b$. Accordingly, processes with specialized arithmetics on two input signals can be defined based on *comb2CT*(f), e.g., the *addition* and *multiplication* processes *addCT*$()$ and *multCT*$()$. Their definitions are excluded from this paper for space reasons. Obviously, *addCT*$(\tilde{s}_1, \tilde{s}_1)$ will get the same result as *multCT*$(2)(\tilde{s}_1)$.

In the time domain[3], a linear and time invariant CT dynamical systems are typically described as ordinary differential equations (ODEs). A general state-space representation (first-order ODEs) of such a vector-valued system with p inputs $\tilde{\mathbf{u}}(t) \in \mathbb{R}^p$, q outputs $\tilde{\mathbf{y}}(t) \in \mathbb{R}^q$, and n state variables $\tilde{\mathbf{x}}(t) \in \mathbb{R}^n$ is as follows.

$$\tilde{\mathbf{x}}'(t) = A \cdot \tilde{\mathbf{x}}(t) + B \cdot \tilde{\mathbf{u}}(t) \tag{4}$$

$$\tilde{\mathbf{y}}(t) = C \cdot \tilde{\mathbf{x}}(t) + D \cdot \tilde{\mathbf{u}}(t) \tag{5}$$

in which A is the $n \times n$ state matrix, B the $n \times p$ input matrix, C the $q \times n$ output matrix, D the $q \times p$ feedthrough matrix, and $\tilde{\mathbf{x}}'(t) = \frac{\mathrm{d}}{\mathrm{d}t}\tilde{\mathbf{x}}(t)$. To solve ODEs in digital computers, the input signals are sampled on demand based on continuous time evolution (leads to DE signals with $\hat{s}(t) = \tilde{s}(t)$), and ODEs are approximated and solved by numerical approaches (to be established as DE processes).

2) DE processes: Based on the recursive operations on DE events, combinatorial DE processes *combDE*(f) and *comb2DE*(f) can be defined similar to CT domains. The computation is a functional mapping from input to output event values, and the partitioning simply takes one event in each computation.

In the following, we use the numerical approximation of ODEs, to demonstrate how numerical filters can be enforced as sequential computations in the DE domain. Typically, *Runge-*

[2]Currying is a functional transformation used to handle functions with multiple arguments by transforming them into a chain of functions that each takes a single argument.

[3]In the frequency domain, transfer functions are used, which can be converted to equivalent ODE representations, e.g., with a Matlab command *tf2ss*.

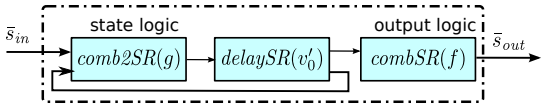


Fig. 3: Moore state machine.

Kutta (RK) methods are used [17], which propagate a solution at each integration step based on CT signals interpolation (DE data events). A classical RK method, with four-stage (RK-4) to estimate the derivative, is as follows.

$$\hat{\mathbf{y}}(t_n) = C \cdot \hat{\mathbf{x}}(t_n) + D \cdot \hat{\mathbf{u}}(t_n) \tag{6}$$

$$\hat{\mathbf{x}}(t_n) = \hat{\mathbf{x}}(t_{n-1}) + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{7}$$

$$\begin{aligned} \textit{where}\quad k_1 &= A \cdot \hat{\mathbf{x}}(t_{n-1}) + B \cdot \hat{\mathbf{u}}(t_{n-1}) \\ k_2 &= A \cdot \hat{\mathbf{x}}(t_{n-1} + \frac{hk_1}{2}) + \frac{B}{2} \cdot (\hat{\mathbf{u}}(t_{n-1}) + \hat{\mathbf{u}}(t_n)) \\ k_3 &= A \cdot \hat{\mathbf{x}}(t_{n-1} + \frac{hk_2}{2}) + \frac{B}{2} \cdot (\hat{\mathbf{u}}(t_{n-1}) + \hat{\mathbf{u}}(t_n)) \\ k_4 &= A \cdot \hat{\mathbf{x}}(t_{n-1} + hk_3) + B \cdot \hat{\mathbf{u}}(t_n) \end{aligned}$$

in which $\hat{\mathbf{u}}(t)$, $\hat{\mathbf{y}}(t)$, and $\hat{\mathbf{x}}(t)$ are the interpolated vector-valued DE signals on $\tilde{\mathbf{u}}(t)$, $\tilde{\mathbf{y}}(t)$, and $\tilde{\mathbf{x}}(t)$ in CT domain (Eq. 4-5) respectively, and $h = t_n - t_{n-1}$ is a fixed integration step without error control. In practice, an adaptive RK method with variable step size can be used (see the implementation in Section III).

3) SR processes: In each evaluation cycle, a SR process consumes one event from the input signals and outputs one event to the output signals. To abstract time information as clock ticks, the synchrony hypothesis states that the computation (reaction) takes no time. There are combinational processes *combNSR*(f) and unit-cycle delay processes *delaySR*(v_0'), in which f specifies the computation function and the value v_0' is the initial output event to defer the input events one cycle. A more complex SR model can be constructed based on the hierarchical composition of them. For instance, a Moore sequential module, which is widely used to specify control logic, is illustrated in Fig. 3. In this graph, g specifies the state function, f the output function, and v_0' the initial state of the Moore machine.

4) SDF processes: SDF models are a subset of dataflow models. In SDF models, each combinatorial process *combNSDF*(f) *consumes* (*produces*) a fixed token rate *from input-side* (*into output-side*) channels, and the computation specified by function f is deterministic. The *delay* on partially ordered signals is simply captured as the initial data tokens in channels. Reading from a channel is blocking, and writing is non-blocking (always succeeds immediately), which implies infinite FIFO sizes. However, based on the given data token rates, the execution of SDF models can be scheduled at compile-time to check whether a deadlock-free schedule with bounded FIFO requirement exists.

III. Implementation and Execution in SystemC

We implement our HetMoC framework in SystemC, inspired by a system level functional modelling style proposed for untimed dataflow models in [18].

Based on event-driven simulation, the synchronization and communication among processes are performed through signals via sc_fifo channels, which connect each producer process to the corresponding consumer process. In HetMoC, only the blocking access operations on sc_fifo channels are supported caused by the bounded FIFO buffers in implementation, i.e., reading from channels is blocking and data tokens are consumed once they are read, and writing to channels is blocking to prevent buffer overflow. Meanwhile, sc_fifo channels are domain-polymorphic, as long as the user-defined data type (of signal events) overrides operator<<. For instance, the DE event can be defined as follows.

Listing 1: DE event definition.

```
1  template<class T>
2  class DEevent {
3   public:
4   DEevent(double t, T v) :
5     timeTag(t), value(v) { }
6   friend ostream& operator<< (ostream& os, DEevent &e) {
7     os << e.timeTag << "\t" << e.value;
8     return os;
9   }
10  double timeTag; T value;
11 };
```

Each process is an instantiated object of a SC_MODULE, in which a SC_THREAD captures the iterative partitioning and computation on input signals. The scheduling order of modules (processes) does not affect the simulation results, and the execution is deterministic. For instance, a *scaleCT* module can be defined as follows. In line 14 and 19, the read and write operations run iteratively to partition the input and output signals as sub-signals. The computation (formally defined in Eq. 2) has been done in line 15-18, which is implemented as lambda functions in the new C++0x (to replace C++03) standard[4].

Listing 2: Implementation of the *scaleCT* module.

```
1  template <class T>
2  SC_MODULE(scaleCT)
3  {
4    sc_fifo_in<CTsubSig> input;
5    sc_fifo_out<CTsubSig> output;
6    SC_HAS_PROCESS(scaleCT);
7
8    scaleCT(sc_module_name N, double k) : sc_module(N), k_(k) {
9      SC_THREAD(proc);
10   }
11
12   void proc() {
13     while (1) {
14       CTsubSig subSig = input.read();
15       Function f = subSig.getFunction();
16       subSig.setFunction([=](double x)->double{
17           return (this->k_ * f(x));
18         });
19       output.write(subSig);
20     }
21   }
22  private:
23   double k_; /* the scaling factor 'k' */
24 };
```

When CT filters are approximated and solved by numerical methods with fixed step sizes, the designer has no information about how the local error behaves. Therefore, numerical methods with error control are more widely accepted in practice.

[4]Another alternative to write lambda expressions is the Boost Lambda Library [19]. However, it shows a performance degradation up to 25% in our experiments and is thus not preferred.

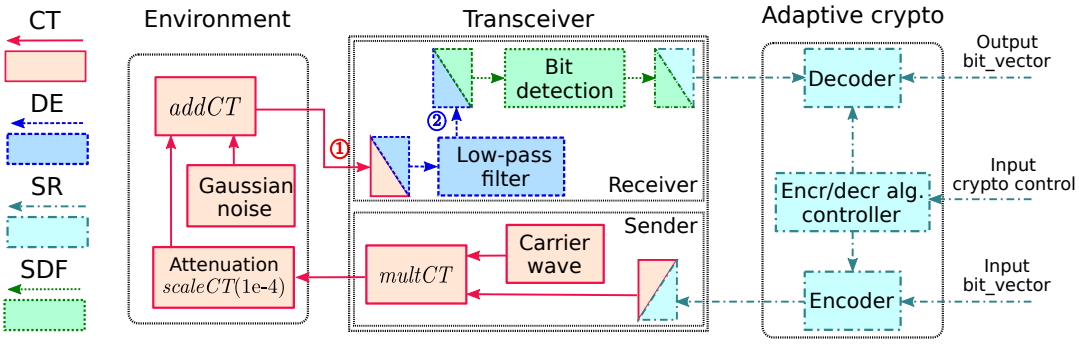


Fig. 5: The synopsis of an ASK transceiver system.

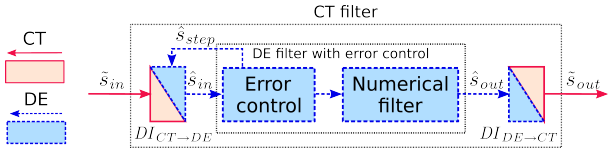


Fig. 4: The digital approximation of a CT filter.

In general, they could adjust step sizes according to the behavior of solutions. The approximation error is calculated systematically as the step size varies in the range of the minimum and maximum step sizes. Each time, the step size is only accepted when the error is smaller than specified, and the step size is increased for the next step. Otherwise, if the error is bigger than specified, the step is redone with reduced step size. Usually, CT filters are specified as either transfer functions or ODEs with continuous dynamics. On the other hand, the implementation is a mixing of CT and DE domains, as illustrated in Fig. 4.

It is an analog filter with CT interfaces ($\tilde{s}_{in}$ and $\tilde{s}_{out}$) from **end users' external view**, but it is based on a numerical DE filter in **internal implementation**. Therefore, the users are not aware of the implementation details and are only required to configure the CT filter coefficients, besides the error tolerance and minimum/maximum step sizes required to specify DE implementation. The DE signals need in numerical analysis is sampled from CT signals via domain interface $DI_{CT \to DE}$. Especially, a feedback signal $\hat{s}_{step}$ from the error control module to $DI_{CT \to DE}$ is used to specify the sampling step size on the input CT signal $\tilde{s}_{in}$. Finally, the output DE signal of the numerical solver is converted back as a CT output signal. Currently, Richardson extrapolation [17] has been used in a RK-4 solver for error control.

Listing 3: Declaration of an *adaptiveCombSR* module.

```
1 template <class S, class T> SC_MODULE(adaptiveCombSR)
2 {
3   sc_fifo_in<S> input;
4   sc_fifo_in<FunctionExt<S,T> > adaptiveFunction;
5   sc_fifo_out<T> output;
6
7   SC_HAS_PROCESS(adaptiveCombSR);
8   ...
9 }
```

For adaptive system with time-varying functionalities (see case study in Section IV), we extend the regular combinatorial processes to model *adaptation*. For clarity, only the declaration of a SR *adaptiveCombSR* module is shown in Listing 3. Besides the input and output data signals, the module has an extra input function stream in line 4. The functions have extended data types and cope with the time-varying mapping from inputs to outputs, assuming that time information has been implicitly associated with SR clock ticks.

IV. Experimental Results

To demonstrate our heterogeneous MoC (HetMoC) framework, we use an adaptive amplitude-shift keying (ASK) transceiver system as a case study, which covers all the heterogeneous MoCs introduced in this paper. For clarity, only a synopsis of the system in heterogeneous MoC domains is illustrated in Fig. 5. As a reference model, it has been modelled in SystemC-AMS (see the following Discussions and Acknowledgements).

The transceiver system consists of an adaptive crypto-module, the transceiver, and the environment. The crypto-module is called *adaptive*, in the sense that the encryption/decryption algorithms may vary during run-time according to an input crypto control signal. In the bottom-right, the input data stream to the system is a synchronous bit_vector, which is encoded by the specified encryption algorithms. Then the bit information is modulated, i.e., multiplied with the amplitude of a CT carrier wave. In the environment, signals may be corrupted by noise and the power will lose in transmission as well, which are resembled by a Gaussian noise and attenuation modules. The input to the receiver goes through a low-pass filter, and the output is used for bit-detection (digitalization). Finally, the received bit stream is decoded. Obviously, to recover the sending bit_vector in the receiver side, it is essential that the encryption/decryption algorithms change *synchronously*. Whenever signals go across different domains, domain interfaces are used.

All experiments are carried out on a Linux 2.6.31 platform with a single-core Pentium-M 2.0GHZ processor and 2GB of DDR2 RAM. The gcc 4.5.0 compiler is used with c++-0x features and -O3 optimization enabled.

In our HetMoC, the solver of the filter has been implemented as *fixed-step* size (HetMoC-F) and *adaptive-step* size (HetMoC-A). In HetMoC-F, all system parameters are specified following the SystemC-AMS reference model. On the other hand, without losing functionality in the transceiver system, HetMoC-A has used an adaptive-step size (in the range of 0.1-10 times to the size in HetMoC-A) according to a pre-specified error tolerance. We use gnuplot for signals

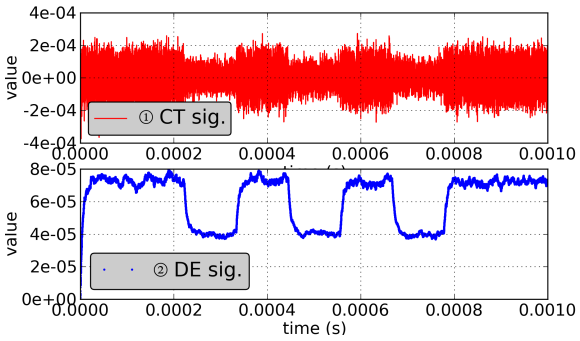


Fig. 6: ① is the input of Receiver and ② is the output of the filter (both are labelled in Fig. 5).

TABLE I: Simulation run time and peak memory.

Metrics	SystemC-AMS	HetMoC [a]	
		HetMoC-F	HetMoC-A
Time (s)	15.8	71.5	14.0
Memory (MB)	20.72	5.54	5.54

[a] Both *fixed-step* (HetMoC-F) and *adaptive-step* (HetMoC-A) algorithms are evaluated.

visualization. In Fig. 6, a period of 1 *ms* (1-bit_vector) of a CT signal (input to the receiver) and a DE signal (output from the low-pass filter) in HetMoC-A simulation are exemplified. For a period of 10 *ms* (10-bit_vector), the simulation run time (s) and peak memory usage (MB) are analyzed when profiling and plotting functions are turned-off, as shown in Table I.

Discussions. The simulation run time of HetMoC is comparable to SystemC-AMS. In several perspectives, the proposed HetMoC framework has shown promising capabilities:

- In SystemC-AMS, both DE and SR domains in the transceiver system can only be captured as T-SDF MoCs, and the simulation of the CT domain is based on a T-SDF simulator as well. In contrast, HetMoC captures all the modelling domains exactly as specified in Fig. 5, and intends to be a formal heterogeneous modelling framework.
- In SystemC-AMS, the solver has been especially optimized for speed for single-input single-input (SISO) first-order filters. In HetMoC, the implementation of the RK solver is more general (MIMO and higher-order), and the simulation accuracy is comparable to Matlab using RK numerical method. Especially, HetMoC-A supports predictable error control with varying step size, and has shown faster simulation speed than in SystemC-AMS.
- The peak memory consumption in HetMoC is less than 27% of the SystemC-AMS approach. Therefore, HetMoC suites large distributed systems modelling and simulation better, given limited computer memory.

V. Conclusion

In this paper, we have presented our HetMoC specification and modelling framework in SystemC for heterogeneous embedded system, which fully integrates all major modelling domains (CT, DE, SR, and untimed) in a seamless way on a sound formal base. In experiments and simulation, HetMoC has shown promising capabilities in several perspectives, e.g., run time and peak memory consumption.

Currently, the fast simulation speed in HetMoC-A has been achieved mostly by adaptive-step size. We plan to further enhance the numerical approximation methods in the solver to improve the simulation speed and efficiency. Also, the HetMoC framework has been developed driven by several case studies, which still needs to be enriched in the future.

Acknowledgements

We thank Markus Damm, Dr. Jan Haase, and Prof. Christoph Grimm for offering the transceiver system reference model in SystemC-AMS, and thank anonymous reviewers for helpful comments on the content of this paper.

References

[1] A. Sangiovanni-Vincentelli, "Quo vadis, SLD? Reasoning about trends and challenges of system-level design," *Proceedings of the IEEE*, vol. 95, no. 3, pp. 467–506, March 2007.

[2] A. Jantsch and I. Sander, "Models of computation and languages for embedded system design," in *IEE Proceedings on Computers and Digital Techniques*, 2005, pp. 114–129.

[3] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming heterogeneity - the Ptolemy approach," in *Proceedings of the IEEE*, 2003, pp. 127–144.

[4] E. A. Lee and H. Zheng, "Leveraging synchronous language principles for heterogeneous modeling and design of embedded systems," in *EMSOFT '07: Proceedings of the 7th ACM & IEEE international conference on Embedded software*. New York, NY, USA: ACM, 2007, pp. 114–123.

[5] F. Herrera and E. Villar, "A framework for embedded system specification under different models of computation in systemc," in *DAC '06: Proceedings of the 43rd annual Design Automation Conference*. New York, NY, USA: ACM, 2006, pp. 911–914.

[6] A. Vachoux, C. Grimm, and K. Einwich, "Extending SystemC to support Mixed Discrete-Continuous System Modeling and Simulation," in *International symposion on circuits and systems 2005 (ISCAS '05)*, Kobe, Japan, 2005.

[7] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, September 1987.

[8] A. Schallenberg, F. Oppenheimer, and W. Nebel, "OSSS+R: Modelling and Simulating Self-Reconfigurable Systems," in *Proceedings - 2006 International Conference on Field Programmable Logic and Applications*, August 2006, pp. 177–182.

[9] M. Damm, J. Haase, C. Grimm, F. Herrera, and E. Villar, "Bridging mocs in systemc specifications of heterogeneous systems," *EURASIP J. Embedded Syst.*, vol. 2008, pp. 1–16, 2008.

[10] H. D. Patel and S. K. Shukla, "Towards a heterogeneous simulation kernel for system level models: A systemc kernel for synchronous data flow models," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 24, no. 8, pp. 1261–1271, 2005.

[11] G. Kahn, "The semantics of a simple language for parallel programming," in *Information Processing '74: Proceedings of the IFIP Congress*, J. L. Rosenfeld, Ed. New York, NY: North-Holland, 1974, pp. 471–475.

[12] 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, December 1998.

[13] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, "The synchronous data flow programming language LUSTRE," *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1305–1320, September 1991.

[14] P. L. Guernic, J. Talpin, and J. L. Lann, "Polychrony for system design," *Journal of Circuits, Systems and Computers. Special Issue on Application Specific Hardware Design*, 2002.

[15] I. Sander and A. Jantsch, "System modeling and transformational design refinement in ForSyDe," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 23, no. 1, pp. 17–32, 2004.

[16] J. Zhu, I. Sander, and A. Jantsch, "Energy efficient streaming applications with guaranteed throughput on MPSoCs," in *Proceedings of the International Conference on Embedded Software (EMSOFT '08)*, Atlanta, USA, October 2008, pp. 119–128.

[17] J. C. Butcher, *Numerical methods for ordinary differential equations*. Wiley, 2003.

[18] T. Grötker, S. Liao, G. Martin, and S. Swan, *System Design with SystemC*. Norwell, MA, USA: Kluwer Academic Publishers, 2002.

[19] Boost C++ libraries, http://www.boost.org/.