A Hybrid Simulation Approach for Fast and Accurate Timing Analysis of Multi-Processor Platforms Considering Communication Resources Conflicts

Sébastien Le Nours, Adam Postula

A Hybrid Simulation Approach for Fast and Accurate Timing Analysis of Multi-Processor Platforms Considering Communication Resources Conflicts

Sébastien Le Nours · **Adam Postula**

Abstract In the early design phase of embedded systems, discrete-event simulation is extensively used to analyse time properties of hardware-software architectures. Improvement of simulation efficiency has become imperative for tackling the ever increasing complexity of multi-processor execution platforms. The fundamental limitation of current discrete-event simulators lies in the time-consuming context switching required in simulation of concurrent processes. In this paper, we present a new simulation approach that reduces the number of events managed by a simulator while preserving timing accuracy of hardware-software architecture models. The proposed simulation approach abstracts the simulated processes by an equivalent executable model which computes the synchronization instants with no involvement of the simulation kernel. To consider concurrent accesses to platform shared resources, a correction technique that adjusts the computed synchronization instants is proposed as well. The proposed simulation approach was experimentally validated with an industrial modeling and simulation framework and we estimated the potential benefits through various case studies. Compared to traditional lock-step simulation approaches, the proposed approach enables significant simulation speed-up with no loss of timing accuracy. A simulation speed-up by a factor of 14.5 was achieved with no loss of timing accuracy through experimentation with a system model made of 20 functions, two processors and shared communication resources. Application of the proposed approach to simulation of a communication receiver model led to a simulation speed-up by a factor of 4 with no loss of timing accuracy. The proposed simulation approach has potential to support automatic generation of efficient system models.

Keywords Timing analysis · Workload models · Discrete-event simulation · Multi-processor platforms

S. Le Nours
University of Nantes, UMR CNRS 6164 IETR
Polytech Nantes, La Chantrerie, 443060 Nantes
E-mail: sebastien.le-nours@univ-nantes.fr

A. Postula
University of Queensland, School of Information Technology and Electrical Engineering, Brisbane, Australia

1 Introduction

More and more embedded systems are designed on multi-processor platforms to satisfy the growing computational demand of real-time applications. Multi-processor platforms are made of heterogeneous components: computation resources (e.g., general purpose processors, specialised processors, dedicated hardware accelerators), communication resources (e.g., buses, interfaces), and storage resources (e.g., shared memories, cache memories). Verification that timing constraints are fully satisfied requires extensive analysis of software running on execution platform. Analysis should be performed early in the design cycle to detect potential design issues and to prevent costly design iterations. However, resource sharing in multi-processor platforms leads to complex interactions among components that makes timing analysis very challenging. It is therefore essential to facilitate creation of high level models of hardware-software architectures that should deliver both reasonable evaluation time and good accuracy.

The emergence of the transaction level modeling (TLM) paradigm has facilitated the description of platform resources at higher levels of abstraction than the traditional register transfer level [5]. This paradigm relies on an explicit separation between computation and communication mechanisms. TLM allows low level details of computation and communication to be hidden and significant simulation speed-up is thus achieved compared to cycle-accurate models, as illustrated in [31] and [12]. In this context, system-level design

approaches have been proposed to analyze application execution onto high level models of platforms [13]. Such approaches extensively make use of discrete-event simulation to analyze the influence of application execution on platform resources under various working scenarios. In discrete-event simulation, simulation events correspond to specific synchronization instants among processes of the system model. The aim of a discrete-event simulation kernel is to correctly manage the time ordered sequence of events among the simulated processes and the advancement of the simulation time. However, synchronizations among processes cause time- consuming context switches in the simulation kernel, that can significantly reduce the simulation speed and lead to unacceptable evaluation time.

In this paper, we introduce a simulation approach that limits the number of simulation events and still preserves the timing accuracy of performance models of hardware-software architectures. The approach we propose consists in abstracting some of the processes of a system model into an equivalent executable model as seen by the simulation kernel. The created executable model incorporates the expressions of the instants when the platform resources are used, and thus the synchronization instants among the abstracted elements are computed during simulation without context switching of the processes. In this paper, we adopt the timed Petri net formalism to formulate the synchronization instants among the abstracted system elements. The originality of this simulation approach lies in the evaluation of the synchronization instants with no involvement of the simulation kernel. However, when the number of events managed by the simulation kernel is reduced, one potential issue is possible degradation of models accuracy. The problem is that access conflicts at platform shared resources are invisible and the delays caused by access conflicts cannot be simulated. We present a simulation technique that preserves the influence of shared resources with a limited number of simulation events. The proposed technique uses knowledge about application and platform to correctly adjust the computed synchronization instants in the case of contention at shared resources. In the scope of this paper, we illustrate the application of the proposed simulation approach to data flow oriented systems with shared communication resources. Such systems are characterised by data-dependent application workloads and simulation is commonly used to evaluate the influence of workload variability and effect of resource sharing on system performance. We implemented and validated the proposed simulation approach and the related techniques using Intel CoFluent Studio modeling framework [15] and SystemC simulation language [14]. Various experiments were led to estimate the achieved simulation speed-up and to evaluate the timing accuracy. Besides, we examined the scalability of the approach and we evaluated the influence of its complexity. A simulation speed-up by a factor of 14.5 was achieved with no loss of timing accuracy for a system model made of 20 functions and two processors. For a system model made of 100 functions the proposed simulation approach led to a simulation speed-up by a factor of 4 with timing accuracy preserved. In this paper, the benefits of the simulation approach are demonstrated through the study of an heterogeneous architecture of a communication receiver implementing the physical layer of the long term evolution (LTE) protocol. Simulation speed-up by a factor of 4 was achieved with no loss of timing accuracy.

This paper is organized as follows. In Section 2, we give an overview of relevant related work. We present the principles of the proposed approach in Section 3. The computation technique of synchronization instants is described in Section 4. The correction technique is detailed in Section 5. We validate our approach and show experimental results in Section 6. We conclude this paper in Section 7.

2 Background and related work

In this section, we first provide a brief review of the main concepts associated with the traditional simulation-based performance evaluation approaches. Next, we give an overview of the main research related to our work.

2.1 Simulation of architecture performance models

Following the Y-chart modeling approach [9], a system model is formed by a combination of an application model and a platform model. In the early design phase, full description of application functionalities is not mandatory and workload models of the application are considered. A workload model expresses the computation and communication loads that an application causes when executed on platform resources. An application workload model is structured as a network of concurrent communicating processes. Processes use abstract primitives to model communication and data processing (e.g., read, write, execute). Different communication policies can be implemented to describe the way data are transferred and the dependencies between processes (e.g., rendezvous policy, FIFO policy with finite or infinite capacity). Fig. 1 depicts a didactic example of a system model. In this example, processes F1 and F2 are allocated to processor P1 and processes F3 and F4 are allocated to processor P2. The platform model could also include dedicated hardware resources to support process execution. The behaviour of each process is given in the right part of Fig. 1. In Fig. 1, relation M2 is communicated through shared communication bus N by interfaces IF1 and IF2. Relation M4 is communicated through shared communication bus N by interfaces IF3 and IF4. The interfaces temporarily store data and manage transfers through the communication bus. A shared

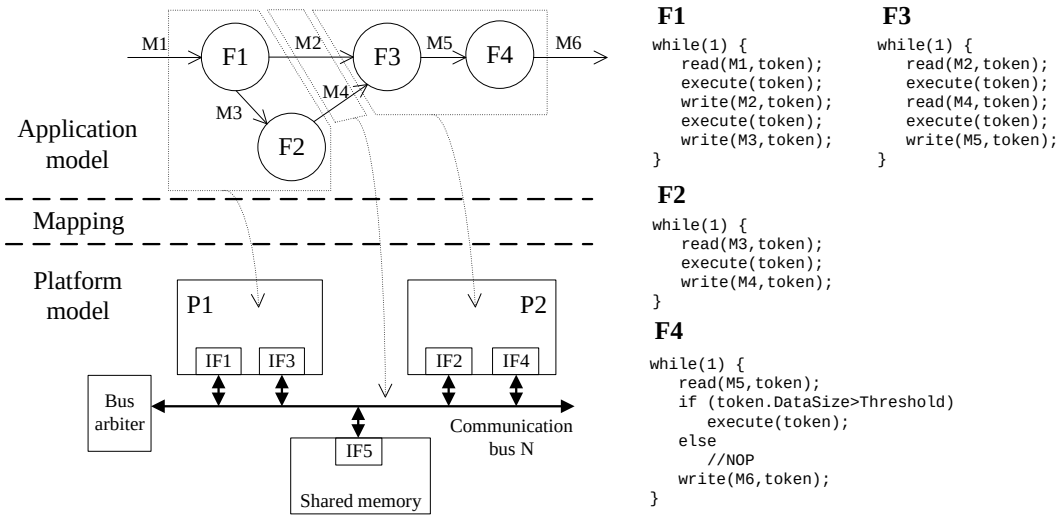


Fig. 1 Example of a system model based on three views: application, platform, and mapping.

memory is also shown. Schedulers are not illustrated in Fig. 1 since in the following we will focus on data flow oriented applications, with non-preemptive static order scheduling.

Academic system-level design approaches such as SCE [10], Sesame [11], SystemCoDesigner [17], and the ones presented in [1] and [19] were proposed to support the creation process of performance models of hardware-software architectures. Industrial frameworks such as Intel CoFluent Studio [15], Space Codesign [33], and Visualsim from Mirabilis Design [25] can also be cited.

In the existing approaches, captured workload models are generated as executable descriptions and then simulated. Different approaches can be considered to generate and simulate system models. In the trace-driven simulation approach [23], a platform model is driven by the traces caused by the application model execution. A trace represents the workload imposed by a communication or a data processing primitive of the application on a platform element. A trace contains information on the communication and computation operations performed by the application process (e.g., size of communicated data). The execution time of each load primitive is approximated as a delay and the values of these delays are variable and data-dependent. Delays are commonly expressed by analytical formulas and typically estimated from measurements on real prototypes or analysis of low level simulations, as illustrated in [19,26]. The evolution of a system model is thus a result of processing the computation and communication loads by the platform resources. During simulation, the simulation kernel schedules the execution of processes and manages the advancement of the simulation time. Simulation events occur at specific instants when processes synchronize between each other. Fig. 2 illustrates some of the events observed during simulation of the example depicted in Fig. 1[1]. In the considered example, delays can be variable and data dependent. As for example, in Fig. 2 the execution time of the `execute` statement of F1 is variable. In Fig. 2, instants $x_i(k)$ denote the intermediate synchronization instants involved in the system model execution. As for example, $x_{WrM2S}(k)$ denotes the k-th instant at which a write operation starts at interface IF1. The instants related to communication through relation M2 are detailed in Fig. 2. For clarity reasons, the instants related to relations M1, M3, M4, M5, and M6 are not detailed. In the following, we denote the instant when an event occurs in the input of the system model by u and the instant when an event occurs in the output of the system model by y. In Fig. 2, $u(k)$ denotes the k-th instant at which F1 receives a message through relation M1 and $y(k)$ denotes the k-th instant at which F4 produces a message through relation M6.

Simulation of system models allows the influence of platform shared resources on application execution to be analyzed. Concurrent accesses at shared resources lead to contentions, and thus additional delays are caused during simulation of the system model. As an illustration of this phenomenon, let us consider that the platform interfaces cannot simultaneously be used for read/write operations and data transfer. In Fig. 2, the rectangle in grey highlights the situation when IF1 is used for data transfer and the write operation is delayed. Contention at shared resources can significantly impact the execution of an application and therefore it must be thoroughly investigated in the system model simulation. As this requires more time-consuming calls to the simulation kernel the overall simulation speed can be substantially decreased.

[1] In Fig. 2 and in the following of the article, the term "simulation time" designates the logical time managed by the simulator.

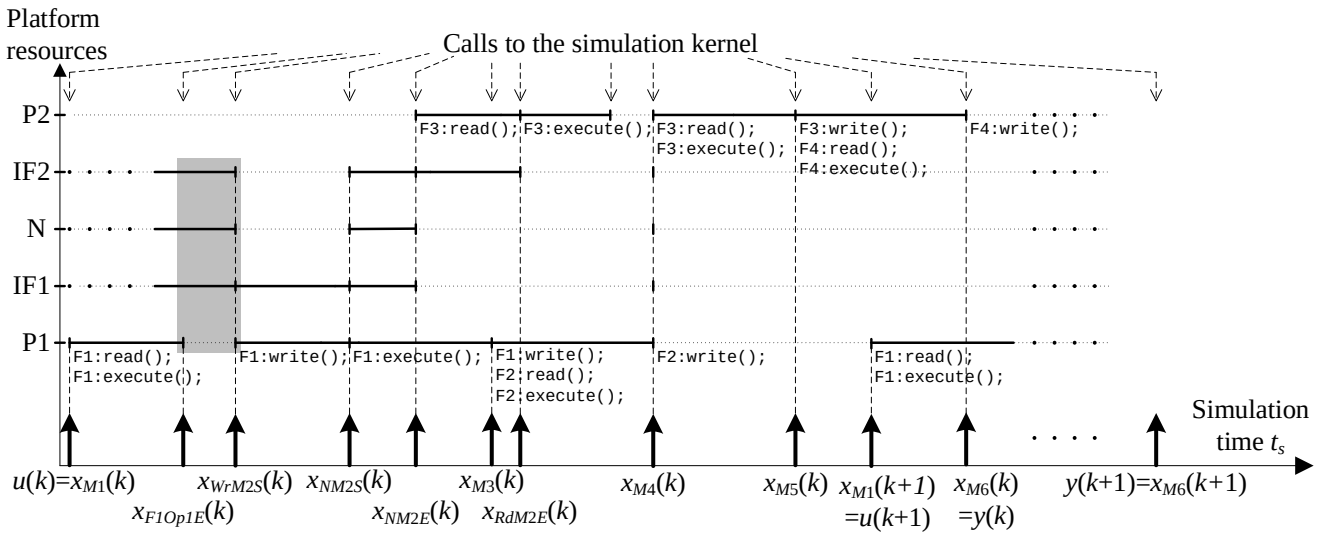


Fig. 2 Discrete-event simulation of the system model with calls to the simulation kernel.

2.2 Improvement of simulation efficiency

Various approaches have been proposed to limit the number of simulation events while maintaining acceptable accuracy of system models.

The SystemC TLM2.0 standard [14] defines two coding styles to allow timing behaviour of architectures to be modeled at different abstraction levels: the approximately-timed coding style (TLM-AT) and the loosely-timed coding style (TLM-LT). Following the TLM-AT coding style, processes of the system model are annotated with specific delay values used in calls to the `wait` function. This coding style leads to a lock-step simulation that provides good accuracy but it causes excessive task switching overheads. To overcome this issue, the TLM-LT coding style supports the temporal decoupling method that allows processes to run ahead in a local time with no use of the simulator. The definition of a global quantum is thus needed to impose an upper limit on the time a process is allowed to run ahead of simulation time. However, this method leads to degraded timing accuracy because delays due to access conflicts at shared resources are not simulated. The approach presented in this paper can be considered as intermediate between these two coding styles. Compared to the TLM-AT coding style, the number of calls to the simulator is decreased by grouping some of the processes of the system model and by computing during simulation the values of delays. Similarly to TLM-LT, the presented approach allows the number of events managed by the simulation kernel to be reduced. A simulation technique is introduced in this paper to consider delays due to potential access conflicts at shared resources.

Bobrek et al. [4] present a hybrid approach that combines discrete-event simulation with analytical models. It aims at decreasing the simulation time of multi-processor platforms with minimum impact on accuracy compared to cycle-accurate models. The application processes are simulated for a period of time by ignoring contention at shared resources. An analytical model is then used to assign time penalties to the competing processes that access to shared resources. The time penalties shift the execution time of the processes running on shared resources. A similar approach is presented by Chen et al. in [7]. It details an analytical model that gives the contention delay under different multi-processor platform configurations and bus utilization rates. This analytical model is used during simulation of the bus model and it prevents full scheduling of events. Compared to these two approaches, our method aims to greatly reduce the number of context switches among processes managed by the simulation kernel.

In [28], Savoiu et al. present an approach to improve simulation performance of SystemC models. This approach consists in restructuring SystemC models to limit the number of calls to the simulator. SystemC models are first converted into Petri nets to be analyzed. Reduction methods are then applied to create an equivalent Petri net but with a limited number of cycles. Finally, an improved SystemC model is obtained from the reduced Petri net. In our approach, we also consider Petri nets as an intermediate representation for workload models. We use this intermediate representation to compute the synchronization instants among the elements of the system model with no use of the simulator. However, we do not consider here the possible reduction of the obtained Petri nets.

In [20], Künzli et al. present a hybrid approach to improve performance evaluation of data flow oriented systems. The presented approach combines analytical models of sys-

tem components and simulation to achieve good compromise between simulation speed and accuracy. Analytical models are based on the real-time calculus (RTC) formalism presented in [6]. This formalism is based on the concept of arrival curves and service curves that respectively characterise workload and processing capabilities of system components. For a given event stream, arrival curves express lower and upper bounds on the number of events arriving in any time interval. In [20], simulation traces are analyzed to determine arrival curves. The curves are then passed to the formal analysis method and the results from the analysis method are transformed into event traces used for simulation. Compared to a simulation model, execution of hybrid models is significantly sped-up because the number of calls to the simulator is limited. Our proposed approach can also be considered as a hybrid approach that combines simulation and formal models. In contrast to [20], our approach does not focus on prediction of best and worst cases.

The result-oriented modeling (ROM) approach has been proposed by Schirner and Dömer in [29,30] to improve the simulation speed-accuracy tradeoff in discrete-event models of communication resources and operating systems. The principle of this approach is to optimistically predict the instants when modeled elements evolve, using available information about resources. The estimated instants are then adapted during simulation in the case of disturbing influences such as pre-emption of shared resources. In our proposed approach, the simulation instants are not estimated but computed during simulation. Compared to ROM, this approach allows more events to be saved and potentially better simulation efficiency.

In [18], simulation constructs are proposed to support creation of accuracy adaptive transaction level models. The created models incorporate multiple levels of timing accuracy and levels can be changed statically at the start of simulation or at runtime. Our proposed approach also considers a reduced number of timing annotations but with no degradation of timing accuracy.

In [34] and [24], simulation techniques are presented that address the timing accuracy problem in temporally decoupled transaction level models. In [34], a central instance keeps track of all transactions that are potentially influenced by other processes and that have not yet synchronized their local time. Once a synchronization point is reached, conflicting transactions are arbitrated and execution times can be revised with a posteriori knowledge. In [24], an analytical timing estimation method is proposed. According to the considered arbitration policy of shared resources, some analytical expressions are defined that give time penalties. Resource usage and resource availability are used to derive the delay formulas. These formulas are then used during simulation to correct the end of each synchronization request. In the same way as in [34,24], our approach performs retroactive timing correction but does not consider TLM-LT models. Besides, we use knowledge about application and platform to reduce the required calls to the simulation kernel.

Razaghi and Gerstlauer present in [27] a conservative simulation approach for multi-core operating system model called automatic timing granularity adjustment (ATGA). ATGA is used to control the advancement of the simulation time and to invoke the simulation kernel whenever a task preemption is required. In predictive mode, the state of running periodic tasks is monitored to determine instants when preemption occurs among tasks. A fallback mode is also implemented that enables asynchronously interrupting long time periods. In our approach, the advancement of the simulation time is done according to computations performed during simulation. Adaptation of the computed instants is required in the case of conflicting accesses at shared resources.

The main thrust of the proposed simulation approach is to reduce the number of required calls to the simulation kernel and to keep accurate the influence of shared resources. The idea of instantaneous computation of synchronization instants was first presented and evaluated in [22] but we did not present how computation could be performed systematically. Besides, the necessity to correct computed instants in the case of shared resources was not considered in [22]. We introduced the idea of correction of synchronization instants in the case of shared resources in [21] but with no details about the implementation of this method. In this article, we present more details of our existing work with the following contributions:

- We present the adoption of the timed Petri net formalism for both the computation and correction of synchronization instants. This formalism is adopted to better justify the feasibility of the computation and correction methods. Some Petri net patterns are introduced to capture the time dependencies among elements of a system model.
- We detail the implementation of the proposed hybrid simulation approach, combining formal and simulation models.
- Based on the timed Petri net formalism, we present the implementation of the correction technique and evaluation of the scalability of this technique.
- We illustrate application of the proposed simulation approach through a case study inspired from communication systems.

3 Principles of the proposed simulation approach

Compared to the existing simulation approaches, the proposed simulation approach uses knowledge about application and platform to limit the number of required calls to the simulation kernel. We consider that some parts of a system model can be abstracted and replaced by an equiva-

lent executable model. This executable model presents the same evolution as the abstracted elements from an external viewpoint, but the number of events managed by the simulation kernel is reduced. The equivalent model incorporates the expressions of the synchronization instants among the abstracted elements[2]. In section 4, we adopt the timed Petri net formalism to express the time dependencies among the abstracted elements and the related synchronization instants. This formal representation is used to compute the synchronization instants during simulation and to determine the execution order between the abstracted processes. This approach allows data-dependent behaviors to be considered (e.g., behavior with delays that depend on exchanged data, behavior with condition on production or reception of data). Data dependencies can be formulated through the created timed Petri net and influence execution of the equivalent executable model. The key point of our approach is that the synchronization instants are instantaneously computed during simulation, i.e. with no involvement of the simulation kernel.

Fig. 3 illustrates the idea of instantaneous computation in comparison with the execution depicted in Fig. 2. In Fig. 3, instants $y(k)$ and $y(k+1)$ are respectively computed at instants $u(k)$ and $u(k+1)$. The computation of intermediate synchronization instants x_i and synchronization instant y at the output of the executable model is denoted by action `Compute_y()`. Compared to the execution depicted in Fig. 2, intermediate synchronization instants x_i are locally computed and stored but no intermediate event is caused during the model execution. The simulation is thus faster because the number of calls to the simulation kernel is reduced, while timing accuracy is preserved. Time dependencies among the synchronization instants can be formulated by the following state equations:

$$X(k) = f(X(k), X(k-1), \ldots, X(k-a), u(k), \ldots, u(k-b)) \quad (1)$$

$$y(k) = g(X(k), \ldots, X(k-c), u(k), \ldots, u(k-d)) \quad (2)$$

$X(k)$ denotes the set of intermediate synchronization instants $x_i(k)$ among the abstracted elements. f reflects the dependencies between current intermediate instants $X(k)$, previous intermediate instants, and synchronization instants at the input of the system model. g reflects the dependencies between current synchronization instants $y(k)$ at the output of the system model, intermediate synchronization instants, and synchronization instants at the input of the system model. For illustration purposes and with no loss of generality, we will only discuss models with one input and one output. In the following, state equations (1) and (2) will be expressed through a timed Petri net.

[2] Besides, application functionalities can be incorporated into the proposed equivalent model.

The presented approach reduces the number of events managed by the simulation kernel. In the case of shared resources, a disturbing influence[3] can cause the computed synchronization instants to become inaccurate because delays due to access conflicts at shared resources are not simulated. We propose to utilize the expression of synchronization instants to detect potential conflicts at shared resources and to correct the erroneous computed instants. This idea is illustrated in Fig. 4, where the shaded rectangles correspond to the intervals of time when a shared resource is used. $x_S(k)$ and $x_E(k)$ correspond to the instants when the usage of the shared resource starts and ends[4]. Instants $x_S(k)$, $x_E(k)$, and $y(k)$ are computed at instant $u(k)$ using the approach illustrated in Fig. 3. In part (a) of Fig. 4, a disturbing influence occurs before $x_S(k)$ and it causes usage of the shared resource. In the case of a shared resource for which simultaneous usage is not possible (i.e. with limited concurrency), computed instants $x_S(k)$, $x_E(k)$, and $y(k)$ become incorrect. Application of the proposed correction technique is shown in part (b) of Fig. 4. The key idea of the correction technique is to re-evaluate state equations (1) and (2) when the disturbing influence occurs. We recall that these equations capture the influence of shared resources and the related delays. The disturbing influence is taken into account straight and it modifies some of the intermediate instants. The state equations are thus re-evaluated with new initial values and internal state $X(k)$ is adjusted accordingly. Output simulation instant $y(k)$ is also corrected. The correction is performed instantaneously with no involvement of the simulation kernel. This correction is denoted in part (b) of Fig. 4 by action `Correct_y()`. In the situation depicted in part (b) of Fig. 4, $x_S(k)$ is corrected and set to the instant when the resource is available. This corrected instant is denoted by $x_S^c(k)$. Based on this corrected value, previously computed instants $x_E(k)$ and $y(k)$ are recalculated through equations (1) and (2), resulting in $x_E^c(k)$ and $y^c(k)$. With this technique, the delays due to contention at shared resources are inserted with no involvement of the simulation kernel. Accurate evaluation can thus be achieved with a reduced set of events. In the scope of this paper, we will illustrate the application of this approach for shared communication resources. Similarly, contention at computation and memory resources could also be addressed.

The presented simulation approach combines thus instantaneous computation and correction of synchronization instants. We successively detail these two techniques in the following of this paper.

[3] Disturbing influence designates any simulation event that causes contention at a shared resource.

[4] As an example, in Fig. 2 interface IF1 is used for data exchange through communication bus N. $x_S(k)$ could represent the k-th instant when a data transfer through node N starts.

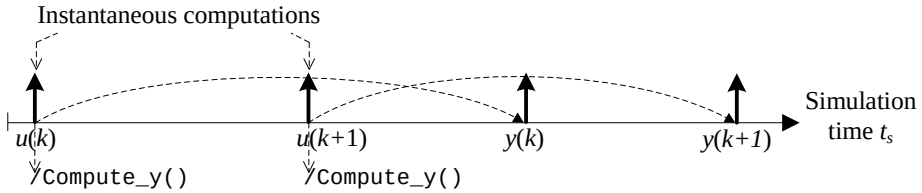


Fig. 3 Proposed hybrid simulation approach with instantaneous computation of synchronization instants.

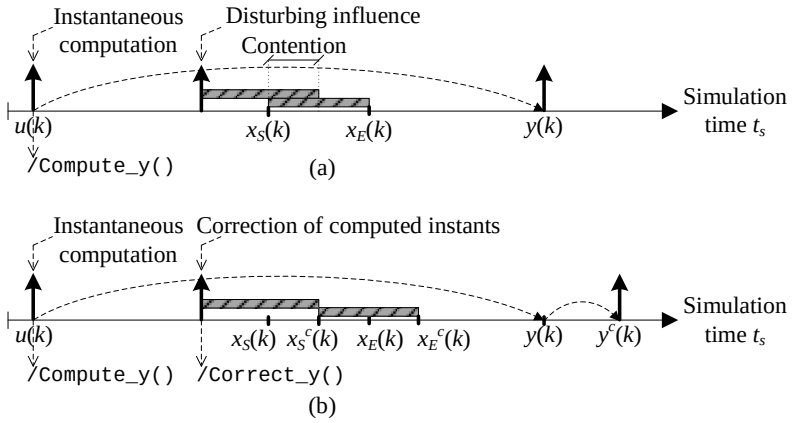


Fig. 4 (a) Simulation with erroneous computed instants in the case of a shared resource, (b) simulation with correction of computed instants.

4 Computation of synchronization instants

We describe the technique for computing synchronization instants. The timed Petri net formalism is adopted to formulate the synchronization instants among the abstracted elements of the system model. We first give some basic concepts related to timed Petri nets. Second, we present the construction of timed Petri nets from system models. Third, we explain the way expression of synchronization instants and executable models are combined.

4.1 Timed Petri nets

Petri nets represent a powerful modeling language that is well adapted for formal description of discrete-event systems. Timed Petri nets represent a timed extension of Petri nets for which time is expressed as minimal durations on the sojourn of tokens in places [2]. A timed Petri net is defined as a tuple $\mathscr{T} = (\mathscr{P}, \mathscr{Q}, {}^\bullet(.), (.)^\bullet, \mu_0, T, \rho)$ where:

- $\mathscr{P} = p_1, \ldots, p_m$ is a finite, non-empty set of places,
- $\mathscr{Q} = q_1, \ldots, q_n$ is a finite, non-empty set of transitions,
- ${}^\bullet(.) \in (\mathbb{N}^{|\mathscr{P}|})^{|\mathscr{Q}|}$ is the backward incidence function,
- $(.)^\bullet \in (\mathbb{N}^{|\mathscr{P}|})^{|\mathscr{Q}|}$ is the forward incidence function,
- $\mu_0 \in \mathbb{N}^{|\mathscr{P}|}$ is the initial marking of the net,
- $T \in \mathbb{R}^{|\mathscr{P}|}$ is the set of holding times of places,
- $\rho \in \mathbb{N}^{|\mathscr{P}|}$ is the set of switching sequences attached to places.

Each transition q is characterised by its backward and forward incidence functions. They indicate the weight of each arc that connects a place and a transition. A marking μ of the net is an element of $\mathbb{N}^{|\mathscr{P}|}$ such that $\mu(p)$ is the number of tokens in place p. Each place has an infinite capacity. Basically, a transition q is enabled if each upstream place contains at least one token. A firing of an enabled transition removes one token from each place of its upstream places and adds one token to each of its downstream places. In a more general way, weights are attached to arcs. A transition is thus enabled if the upstream places contain at least the number of tokens given by the weight of the connecting arcs. Similarly, after the firing of a transition, a downstream place receives the number of tokens given by the weight of the connecting arc. In the following, when not indicated, the weight of the arc is equal to 1. The holding time of a place is the time a token must spend in the place before contributing to the enabling of the downstream transitions. Holding times can depend on the index of the firing. Each place p that has several downstream transitions receives a switching sequence $\rho(p)$. It defines the transition to which the token must be routed. Only tokens such that $\rho(p) = q$ should be taken into account by q. Fig. 5 illustrates a simple timed Petri net with seven places and six transitions. In the situation depicted in Fig. 5, tokens are initially set into places p_4 and p_6.

We denote $x_j(k)$ $j = 1, \ldots, |\mathscr{Q}|, k \geq 0$, as the instant when transition q_j is enabled for the k-th time. Relationships between transition instants can basically be expressed using two operators: addition and maximization. Addition expresses a time lag according to an holding time. Maximization re-

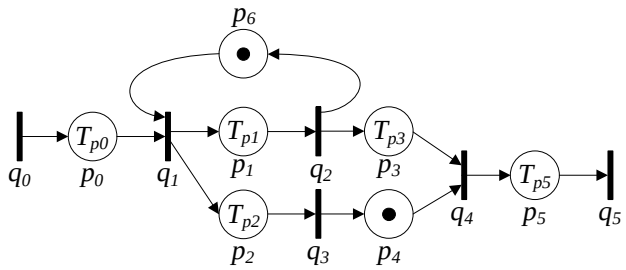


Fig. 5 Example of a timed Petri net with seven places and six transitions.

flects the effect of synchronization. With the initial marking depicted in Fig. 5, the transition instants are given as follows:

$$x_1(k) = max(T_{p0}(k) + x_0(k), x_2(k-1))$$
$$x_2(k) = T_{p1}(k) + x_1(k)$$
$$x_3(k) = T_{p2}(k) + x_1(k)$$
$$x_4(k) = max(T_{p3}(k) + x_2(k), x_3(k-1))$$
$$x_5(k) = T_{p5}(k) + x_4(k)$$

Considering $X(k)$ as the vector formed by transition instants $x_j(k)$ of the timed Petri net, the transition instants can be expressed through state equations similar to (1) and (2). The state equations are obtained by considering input transition instant $u(k)$ as $x_0(k)$ and output transition instant $y(k)$ as $x_5(k)$[5]. Extensive presentation of the theoretical framework that justifies and analyzes such state equations for timed Petri nets can be found in [2]. In the scope of this paper, we adopt the state equation notation to conveniently explain the developed simulation techniques.

4.2 Construction of a timed Petri net

The timed Petri net notation is adopted here to formulate the time dependencies among the elements of a system model and the related synchronization instants. A timed Petri net is constructed by successively considering the application description, the mapping, and the platform constraints. Elementary timed Petri net patterns must be defined for each statement of the system model. The timed Petri net related to the system model is then obtained by connecting the patterns together. First, connection between patterns is done considering the behavioral description of each process and the structural description of the application. Second, the timed Petri net associated with the system model is formed considering the mapping and platform constraints. A similar approach was adopted in [32] considering the enhanced function flow block diagram (EFFBD) notation. In this paper, we

[5] In this simple example, we obtain state equations with $a = 1$, $b = c = d = 0$. Besides, $y(k)$ does not depend on $u(k)$ in equation (2).

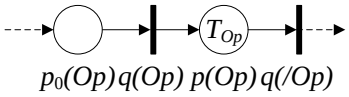


Fig. 6 Petri net pattern of the computation statement.

do not use a specific notation for the system model and we give patterns for some elementary modeling statements.

As previously illustrated, application workload models are made of three categories of modeling statements:

- the computation statement (e.g., execute) models the data processing in application processes,
- the communication statements (e.g., read, write) model the communication protocols among processes of the application,
- the control statements (e.g., alternative, iteration) describe the control flow of each elementary process of the application.

We associate a Petri net pattern with each statement of the application model. Each pattern n can formally be defined by a Petri net $\mathscr{T}^n$. The Petri net related to an application model is obtained by connecting each elementary pattern n. The timed Petri net associated with the system description is then formed by considering the mapping and the platform constraints. First, delays due to execution of application on platform resources are taken into account. Holding times are added to places of the Petri net to represent computation and communication delays. Second, the limited concurrency of platform resources is considered by adding supplementary places and arcs to the created Petri net. These places and arcs cause additional conditions to enable Petri net transitions. In the following, we present patterns with time constraints. These patterns correspond thus to elements of the application model that are allocated to computation or communication resources. Each pattern begins with an entry place and ends with a transition. Dashed lines represent possible connections with other patterns.

The pattern associated with the computation statement is represented in Fig. 6. Transitions $q(Op)$ and $q(\overline{Op})$ denote the instants at which the computation starts and ends. Place $p(Op)$ denotes the execution of the computation whereas $p_0(Op)$ represents the condition before the beginning of the computation. Duration T_{Op} represents the execution time of the operation for the considered computation resource. T_{Op} can be variable and data dependent and it can be expressed by an analytical formula.

According to the communication protocol between processes of the application, different patterns can be associated with the communication statements. Part (a) of Fig. 7 illustrates the pattern for the rendezvous communication protocol. Transition $q(\overline{Rd})$ represents the instant when the writer

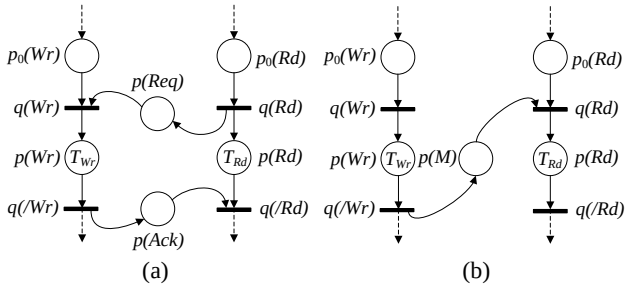


Fig. 7 Petri net pattern of the communication statements: (a) rendezvous protocol, (b) FIFO protocol with infinite capacity.

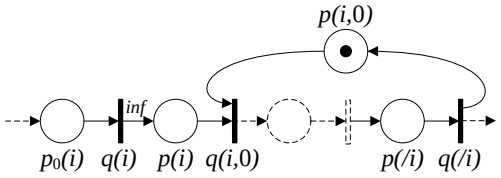


Fig. 8 Petri net pattern of the infinite loop statement.

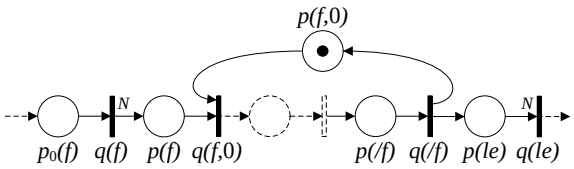


Fig. 9 Petri net pattern of the finite loop statement.

and the reader are synchronized and the data transfer is done. Part (b) of Fig. 7 illustrates the pattern for the FIFO protocol with infinite capacity. Transitions $q(Wr)$ and $q(Rd)$ denote the instants at which write and read operations start. T_{Wr} and T_{Rd} represent the duration of write and read operations for the considered computation resource. They can also be variable and data dependent and they can be expressed by an analytical formula.

The infinite iteration statement is depicted in Fig. 8. Transitions $q(i,0)$ and $q(\bar{i})$ denote the instants when one iteration starts and ends. Place $p(i,0)$ is needed because only one iteration can take place at the same time. Symbol inf means that an infinite number of tokens is added to place $p(i)$ when $q(i)$ is enabled.

The pattern associated with the finite iteration statement is represented in Fig. 9. Symbol N represents the number of iterations of the loop. Transition $q(le)$ denotes the instant when N iterations of the loop have been performed. Place $p(f,0)$ indicates that only one iteration can take place at the same time.

The alternative statement indicates which set of instructions is executed, according to conditions. The pattern associated with the alternative statement is represented in Fig. 10. The switching sequence $\rho(p(a))$ indicates the selection between transitions $q(a1,0)$ and $q(a2,0)$ when a token is into place $p(a)$.

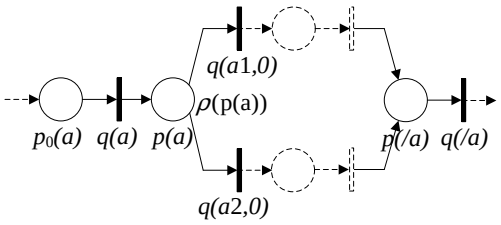


Fig. 10 Petri net pattern of alternative statement.

We illustrate now the modeling of a shared resource with limited concurrency. We consider here the situation where communication between processes is done through two interfaces and a communication bus. As in the example of Fig. 1, we assume that data transfers and read/write operations cannot be performed simultaneously. Fig. 11 illustrates the pattern in the case of a direct data transfer with no usage of a shared memory. The considered bus arbitration policy corresponds to a First Come First Serve (FCFS) policy but other arbitration policies could be modeled. T_{Wr} represents the duration of the write operation into the interface, T_{Rd} represents the duration of the read operation from the interface. T_N represents the transfer duration through communication bus N. Places $p(IF1)$ and $p(IF2)$ model the limited concurrency of interfaces IF1 and IF2. In Fig. 11, once transition $q(Wr)$ is enabled, tokens that are stored in $p(M)$ cannot be processed until $q(\overline{Wr})$ is enabled. A similar pattern could be built in the case of a computation resource with limited concurrency.

The presented patterns are used to create a timed Petri net in association with the elements of a system model. As an illustration, Fig. 12 represents the timed Petri net related to the example depicted in Fig. 1. The depicted Petri net is obtained by connecting together the patterns related to the communication, computation, and control statements. In Fig. 12, the FIFO communication protocol with infinite capacity is considered. The communications through relations M2, M3, M4, and M5 are detailed. T_{NM2} and T_{NM4} represent the transfer durations through communication bus N. In Fig. 12, all the considered delays can be variable and data dependent. Places $p(IF2)$ and $p(IF4)$ express that data transfer and read operation cannot be performed simultaneously by interfaces IF2 and IF4. Places $p(IF1)$ and $p(IF3)$ and related arcs are not depicted in Fig. 12 for clarity reasons. In Fig. 12, a switching sequence, denoted by $\rho(p(a))$, is attached to place $p(a)$. It models the alternative statement used in process F4 of Fig. 1. The created timed Petri net highlights the time dependencies among the elements of the system model and the related synchronization instants. Especially, it gives the time dependencies between the transition instants that are related to the input and the output of the system model (i.e. transitions $q(M1)$ and $q(M6)$ in Fig. 12).

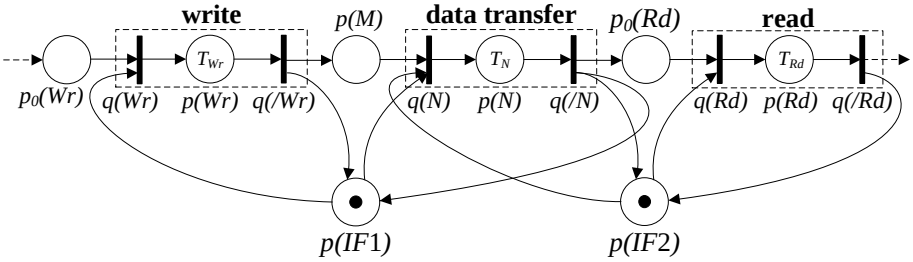


Fig. 11 Petri net pattern in the case of a communication through two interfaces and a communication bus.

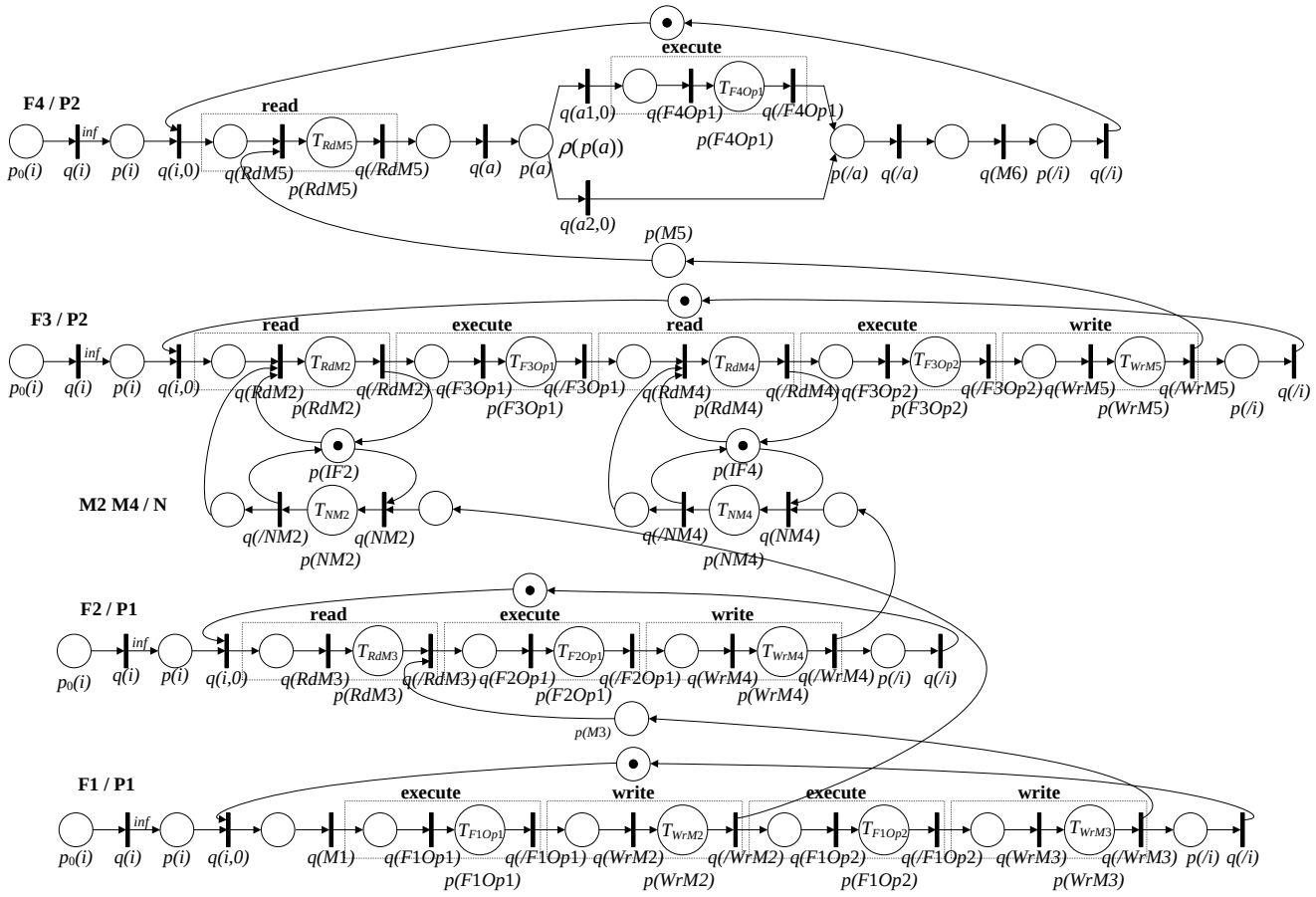


Fig. 12 Timed Petri net associated with the architecture model depicted in Fig. 1.

We use this formal model to compute the synchronization instants among the elements of the system model.

4.3 Combination between formal and executable models

The dependencies among transition instants can be practically captured through a directed graph, that we call a temporal dependency graph. A temporal dependency graph expresses the dependencies among the transition instants of the created timed Petri net. It can also be seen as a convenient way to express the evolution of $X(k)$ and $y(k)$ and to implement state equations (1) and (2). The nodes of a temporal dependency graph correspond to the transition instants of the created timed Petri net. The arcs give the dependencies among the transition instants. The weights associated with the arcs correspond to the delays between the transition instants. In the following, storage of the computed transition instants is done through a one dimensional array denoted by G, which size is denoted by k_G. G aims to temporarily store the set of intermediate instants that are involved in the computation of synchronization instants $X(k)$ and $y(k)$. As an illustration, Fig. 13 depicts a part of the temporal dependency graph related to the timed Petri net of Fig. 12 and the related array G. In this example, the complete temporal de-

pendency graph contains 46 nodes. The transition instants are successively computed by traversing a temporal dependency graph. Traversing of the temporal dependency graph is performed in accordance with the created timed Petri net. Some of the computations performed when traversing the temporal dependency graph depicted in Fig. 13 are given below:

$u(k) = max(x_{M1}(k), x_{M3WrE}(k-1));$
$x_{F1Op1S}(k) = x_{M1}(k);$
$x_{F1Op1E}(k) = x_{F1Op1S}(k) + T_{F1Op1}(k));$
$x_{aS}(k) = x_{M5RdE}(k);$

if (token.DataSize $>$ Threshold){
$x_{a1}(k) = x_{aS}(k);$
$x_{F4Op1S}(k) = x_{a1}(k);$
$x_{F4Op1E}(k) = x_{F4Op1S}(k) + T_{F4Op1S}(k);$
$x_{aE}(k) = x_{F4Op1E}(k);$
}

else{
$x_{a2}(k) = x_{aS}(k);$
$x_{aE}(k) = x_{a2}(k);$
}
$y(k) = x_{M6}(k) = x_{aE}(k);$

In the proposed approach, the system model is simulated through an executable model that presents the same evolution as the abstracted processes from the external viewpoint. This executable model uses the synchronization instants at the input of the system model to compute the intermediate synchronization instants and the synchronization instants at the output of the system model. The computation uses the constructed temporal dependency graph and related array G. The evolution of the executable model mainly depends on two categories of time interval:

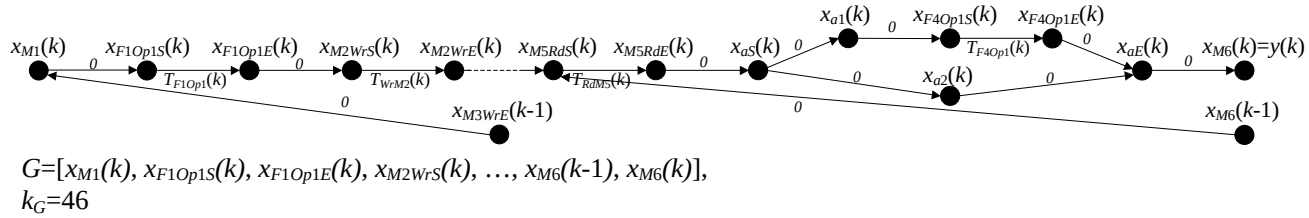
- the minimal interval of time before an output is produced, denoted by T_y,
- the minimal interval of time before an input can be consumed, denoted by T_u.

Fig. 14 gives the structural and behavioural description of the executable model for the example of Fig. 1. Two processes are handled by the simulation kernel. The input process, denoted by Reception, receives input data and computes the synchronization instants through action `Compute_y()`. This action uses array G to locally compute and store the synchronization instants. The computed synchronization instants are then copied in a variable denoted by XS. XS is a two dimensional array that temporarily stores the computed synchronization instants that have not yet been achieved during simulation. The dimensions of array XS have ranges 0 to $k_G - 1$, and 0 to $k_S - 1$. k_S sets of synchronization instants can thus be stored in data structure XS. The ouput process, denoted by Emission, is activated each time a new output instant has been computed. The k-th output data is produced at instant $y(k)$. In Fig. 14, the k-th input data can be received when instant $x_{M3WrE}(k-1)$ is achieved. Intervals of time T_u and T_y correspond to the duration of state s_1 in processes Reception and Emission. The k-th set of synchronization instants stored in XS is released when $y(k)$ is achieved. This is denoted by action `Release_XS()` in Fig. 14. The executable model depicted in Fig. 14 can be adopted for system models with one input and one output. It can be extended to consider multiple inputs and multiple outputs.

The upper part of Fig. 15 illustrates the execution of the equivalent model over the simulation time. When a new data is received through relation M1 at instant $x_{M1}(k)$, the value of output evolution instant $x_{M6}(k)$ is instantaneously computed and stored. Intervals of time T_u and T_y are determined through action `Compute_y()`. The evolution over the simulation time of the equivalent executable model only depends on computed values T_y and T_u. The intermediate synchronization instants are also computed at instant $x_{M1}(k)$. The lower part of Fig. 15 represents the computed intermediate synchronization instants. This observation is performed using a local time called observation time, denoted by t_o. The evolution of the resource usage between $x_{M1}(k)$ and $x_{M6}(k)$ is done without using the simulator. In Fig. 15, the correction technique is not yet considered.

5 Correction of computed synchronization instants

The presented computation technique allows the number of events managed by the simulation kernel to be reduced. However, the computed synchronization instants potentially need to be updated to correctly reflect the influence of shared resources. Fig. 16 illustrates such a potential issue. It focuses on the situation when a communication node and an interface are used for data transfer. In part (a) of Fig. 16, instants x_{RdS} and x_{RdE} correspond to the instants when an interface read operation starts and ends. The shaded rectangle represents the interval of time when the interface is used for reading. Instants $x_{RdS}(k)$, $x_{RdE}(k)$, and $y(k)$ are computed at instant $u(k)$ using the previously presented approach. In part (b) of Fig. 16, the instants at which a new data transfer starts and ends are computed at instant $u(k+1)$. They are denoted by $x_{NMS}(k+1)$ and $x_{NME}(k+1)$. In this situation, the interface is thus simultaneously used to transfer data through node N and to provide the previously received data. In the case of an interface with limited concurrency, these two operations cannot be performed simultaneously and the computed instants are erroneous. The application of the proposed correction technique is illustrated in part (c) of Fig. 16. Once computed, instants $x_{NMS}(k+1)$ and



$G=[x_{M1}(k), x_{F1Op1S}(k), x_{F1Op1E}(k), x_{M2WrS}(k), \ldots, x_{M6}(k\text{-}1), x_{M6}(k)]$,
$k_G=46$

Fig. 13 Temporal dependency graph used to compute transition instants.

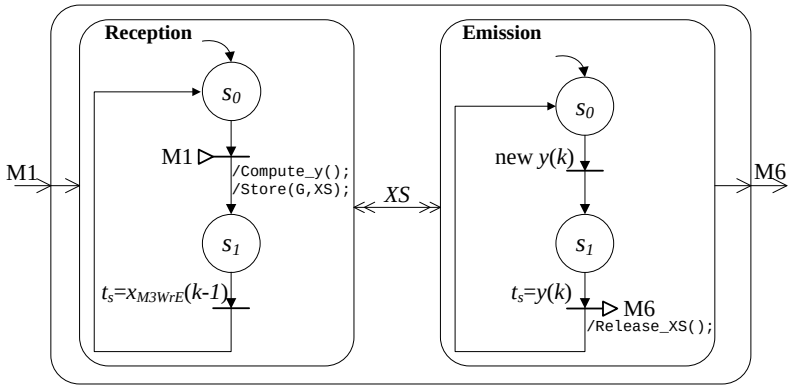


Fig. 14 Structural and behavioural description of the equivalent executable model.

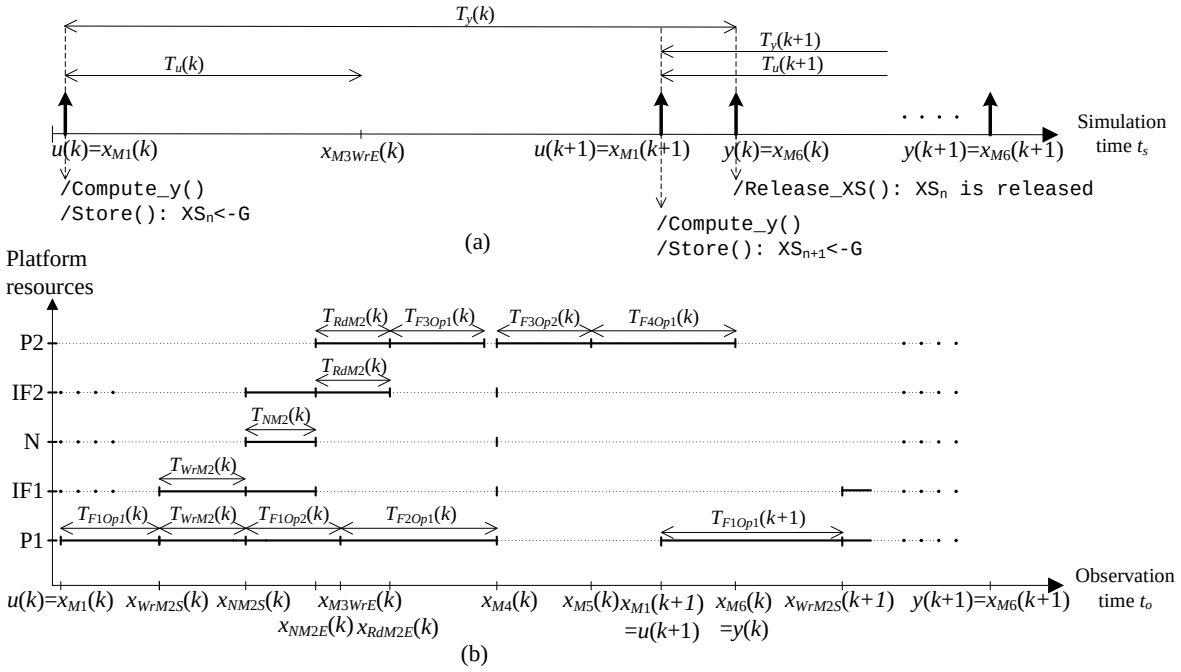


Fig. 15 Execution of the system model using instantaneous computation of evolution instants, evolution is over the simulation time (a) and the observation time (b).

$x_{NME}(k+1)$ are compared to the previously computed instants and a contention at the interface is detected. Instant $x_{RdS}(k)$ is thus corrected accordingly. The previously computed instants that depend on $x_{RdS}(k)$ are then recomputed. The recomputation is done using the expression of the synchronization instants. In Fig. 16, these operations are denoted by action `Correct_y()`. The corrected synchronization instants are denoted by $x^c_{RdS}(k)$, $x^c_{RdE}(k)$, and $y^c(k)$.

The correction technique is implemented in the equivalent executable model presented in Fig. 14. The correction technique is used during simulation to evaluate a new time the previously computed instants. This can be perceived as

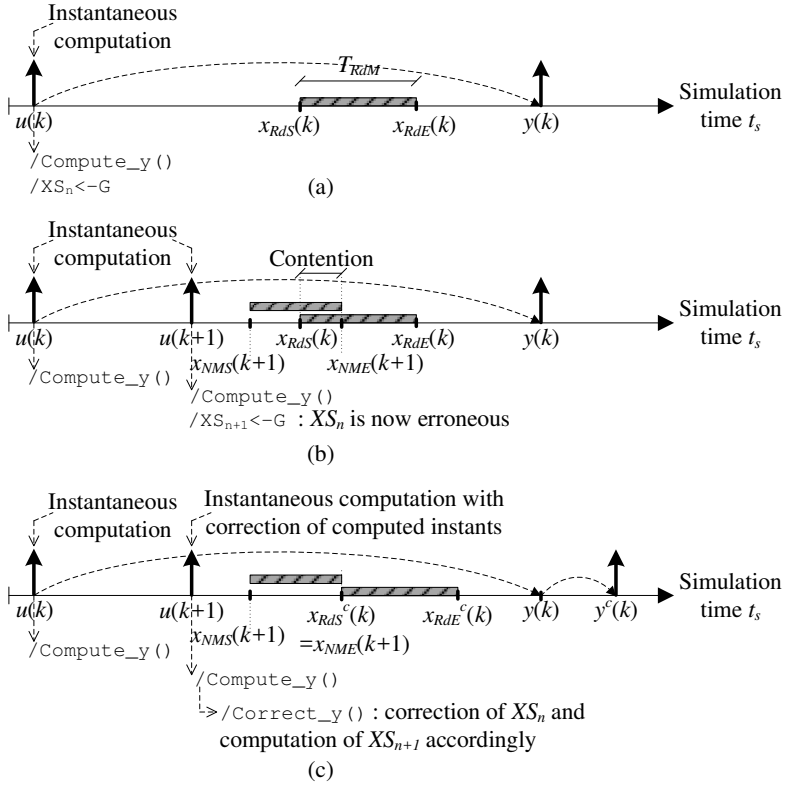


Fig. 16 (a) Equivalent model execution with usage of a shared resource, (b) disturbing influence in the case of a shared resource, (c) equivalent model execution with correction of computed instants.

computing state equations (1) and (2) with new intermediate instants. When a contention is detected, action `Correct_y()` adjusts the previously computed instants stored in data structure XS. We explain here the use of data structures G and XS to correct the previously computed instants. As illustrated in Fig. 16, we consider here that $n+1$ sets of synchronization instants are stored in XS when a correction occurs. We denote by $XS_{m,n}$ the previously computed instant that must be corrected. We denote by G_S and G_E the newly computed instants when usage of a shared resource starts and ends. The basic steps of the correction process are given as follows:

1. (Contention detection.) If $G_S < XS_{m,n} < G_E$, set $XS_{m,n} \leftarrow G_E$.
2. (Temporary saving of data structure G.) For each synchronization instant i from 1 to p, set $D_i \leftarrow G_i$.
3. (Initialize data structure G.) For each synchronization instant i from 1 to p, set $G_i \leftarrow XS_{i,n}$.
4. (Recomputation of synchronization instants.) For each synchronization instant i from $p+1$ to k_G:
   (a) Compute G_i.
   (b) Set $XS_{i,n} \leftarrow G_i$.
5. (Recovery of data structure G.) For each synchronization instant i from 1 to p, set $G_i \leftarrow D_i$.
6. (Return data structure XS.) Return XS_n.

In step (1), when a contention is detected, previously computed instant $XS_{m,n}$ is corrected and set to G_E. In step (2), the values stored in data structure G are temporarily saved because G is used for recomputation. In step (3), G is initialized with some of the previously computed instants. We denote by p the number of synchronization instants that must be loaded in data structure G before recomputation. In step (4), the synchronization instants are successively recomputed and stored in data structure XS. Recomputation is still done by traversing the defined temporal dependency graph. Finally, data structure G is recovered in step (5) to allow the computation of new intermediate synchronization instants. In that case, the complexity of the correction technique is related to the number of synchronization instants to be recomputed.

In the presented situation, only one set of synchronization instants stored in XS is modified. More generally, the correction of previously computed instants can concern various sets of synchronization instants. This situation is illustrated in Fig. 17. In the upper part of Fig. 17, a disturbing influence causes $x_S(k-1)$, $x_E(k-1)$, $x_S(k)$, and $x_E(k)$ to become erroneous. The correction of the computed instants is considered in the lower part of Fig. 17. The correction can be perceived here as using state equations (1) and (2) to successively compute $X(k-1)$, $y(k-1)$, and then $X(k)$,

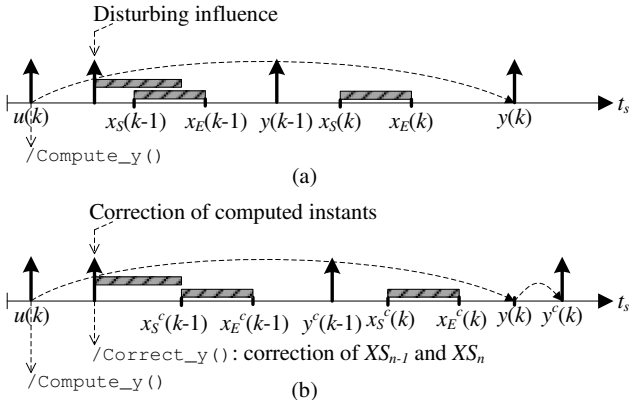


Fig. 17 (a) Disturbing influence in the case of a shared resource, (b) equivalent model execution with correction of multiple sets of synchronization instants.

$y(k)$. During the correction process, steps (3) and (4) must be performed iteratively to correct the required sets of synchronization instants stored in XS. Therefore, the complexity of the correction technique depends on the number of synchronization instants and the number of sets of synchronization instants to correct. The influence of the complexity of the correction technique is evaluated in next section.

6 Experiments

Validation of the proposed simulation approach is first presented. We then estimate the influence of the complexity of the proposed approach. Finally, the benefits of the simulation approach are highlighted through a case study.

6.1 Validation of the simulation approach

Validation of the proposed simulation approach first concerns the Petri net patterns used for computation of synchronization instants. We considered the industrial modeling and simulation framework Intel CoFluent Studio [15] to capture system models and to compare achieved simulation results when using the proposed simulation approach. Using this framework, system models are typically described through three views: application, platform, and mapping views. Once captured, system models are generated as SystemC descriptions and then their executions can be analyzed. We established Petri net patterns for the main modeling statements supported by Intel CoFluent Studio. The equivalent model depicted in Fig. 14 was captured to allow simulation with the proposed approach. The implementation of actions `Compute_y()` and `Correct_y()` corresponded to C++ code developed to compute and correct synchronization instants. The Petri net patterns were validated by comparing the achieved simulation results for various system models. Models were simulated with randomly spaced input stimuli and random values of delays for computation and communication statements. Models were run on a 2.80 GHz Intel-E5 machine with 8 GBytes RAM.

We consider here the didactic example of Fig. 1 to illustrate the benefits of the proposed approach. In this example, the communication interfaces are the only shared resources of the platform model with limited concurrency, i.e. read/write operations and data transfers can not be performed simultaneously by each interface. First, we used Intel CoFluent Studio to capture the system model of Fig. 1. The achieved model was executed using the trace-driven simulation adopted in Intel CoFluent Studio and it was considered as the reference execution. We then used this framework to implement models following the proposed approach. In the considered example, processes F1, F2, F3, and F4 mapped on platform resources were abstracted in an equivalent model. The model of Fig. 14 was used in three different scenarios. First, an execution using instantaneous computation of synchronization instants was considered and no correction of synchronization instants was implemented. Second, an execution using instantaneous computation of synchronization instants was considered with the proposed correction technique. The correction technique was only used to adjust the instants when concurrent accesses to interfaces IF1 and IF2 were detected (i.e., data transfers through relation M2). The third execution was considered with correction of the computed instants related to data transfers through relations M2 and M4. The experimental setup consisted of periodically generated data and communication and computation delays were randomly set for each data processed by the system model. We evaluated the accuracy of the approach by comparing the simulation instants obtained for 10000 generated data. The accuracy of models was evaluated for different occupation rates of the interfaces. Higher occupation rates caused more conflicts at shared resources and more corrections were needed. For each model, the error rate was measured as the number of erroneous synchronization instants over all computed synchronization instants. Fig. 18 presents the achieved accuracy with and without application of the proposed correction technique. As expected, when no correction was performed, the model accuracy degraded because the delays due to access conflicts were not simulated. When the correction technique was used, the errors were compensated. In the case of correction of instants related to data transfers through M2 and M4, full error correction was achieved. In this situation, full error correction was achieved whatever was the value of the occupation rate. When the occupation rate of the interfaces increased, different sets of synchronization instants were corrected. In Fig. 18, observation (1) corresponds to a situation where two sets of synchronization instants were corrected. Observations (2) and (3) respectively correspond to situations where up to

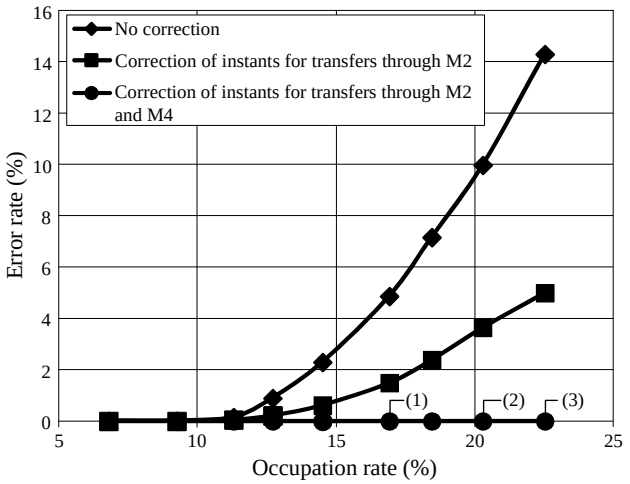


Fig. 18 Accuracy of simulation models according to the occupation rate of the interfaces.

Table 1 Measurement of achieved simulation duration

Execution	Simulation duration (*s*)
Trace-driven simulation	25
State-based model execution with no correction	1.4
State-based model execution with correction	1.4

three and five sets of synchronization instants were corrected during simulation.

The durations of the simulations were measured for 10000 generated data. The considered occupation rate was set at 14.5% which corresponds to the situation where two sets of synchronization instants are corrected. Each model was executed ten times and no significant variation of the execution time was observed. The average durations are given in Table 1. As expected, the reduction of the number of simulation events resulted in shorter simulation times. The achieved simulation speed-up is evaluated to 17.86 with no loss of timing accuracy. Besides, we observe that in the considered experiment the correction technique has no significant influence on the model execution time.

6.2 Influence of the complexity of the approach

In the considered approach, the number of synchronization instants that are computed, and thus the size of data structure *G*, depends on the number of processes of the application that are abstracted. The complexity of the approach strongly depends on the number of elements that are abstracted. To evaluate this relationship, we considered different system models with the same platform model, but with a varying number of processes. The organisation of the studied systems is illustrated in Fig. 19. Architecture models with different sets of processes were successively considered and

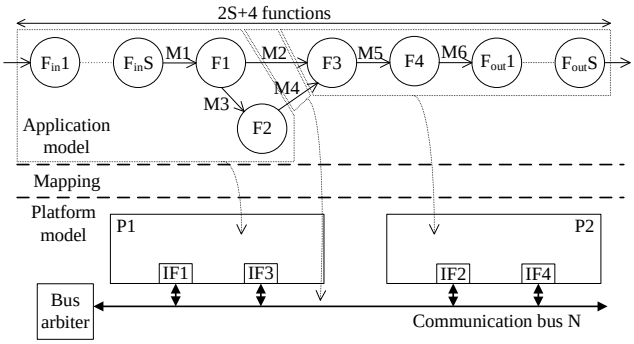


Fig. 19 System model with varying number of processes.

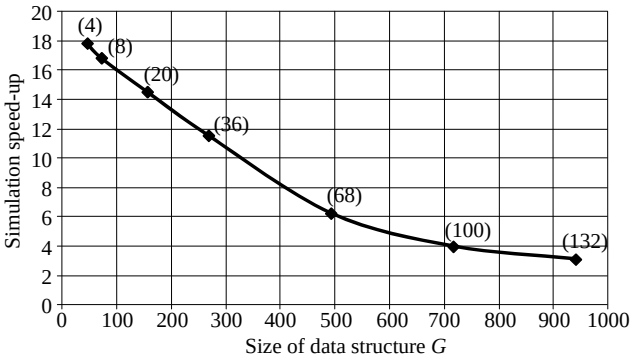


Fig. 20 Influence of the complexity of computation and correction techniques on the achieved simulation speed-up.

captured using Intel Cofluent Studio. The interfaces were still considered as the only shared resources with limited concurrency. The proposed approach was adopted for each system model and simulation efficiency was evaluated. In every case, timing accuracy was still preserved by application of the proposed approach. The simulation runtime was measured with the same conditions as in the previous experiment. Fig. 20 gives the mean measured simulation speed-up for the different system models. The achieved simulation speed-up is given according to the size of data structure *G* used to compute and correct synchronization instants. The number of processes that were abstracted using the proposed approach is given in parenthesis. The previous observation considered an application with four processes, the size of data structure *G* was 46, and the simulation speed-up was 17.86. As expected, when the number of abstracted processes increases the computation and correction techniques get more influence on the simulation duration. As for example, in the situation where 20 processes were abstracted a simulation speed-up of 14.5 was achieved with no loss of timing accuracy. In the situation where 100 processes were abstracted, the size of data structure *G* was more than 700. The achieved simulation speed-up was about 4 with still the same timing accuracy.

This experiment illustrates the influence of the complexity of the computation and correction techniques. Various

factors influence the achievable speed-up. One factor is about the amount of simulation events that can be saved by abstracting some elements of a system model. Therefore, this approach can be efficiently applied to abstract system models with long event streams and many communications between processes. Application of this simulation approach requires good understanding about the abstracted modeling statements. It represents thus a potential solution when existing performance models are reused and combined to consider systems on a larger scale.

6.3 Case study: modeling and simulation of a communication receiver architecture

The case study concerns the analysis of a communication receiver implementing part of the LTE physical layer. The LTE protocol is adopted for fourth generation of mobile radio access [8]. The baseband architecture demands high computational complexity under real-time constraints and multiprocessor implementation is required [16]. This protocol supports high flexibility to answer varying user demands. The studied architecture is depicted in Fig. 21. The application depicted in the upper part of Fig. 21 represents a single input single output (SISO) configuration of a LTE receiver. The baseband functions of the receiver are OFDM demodulation, channel estimation, equalization, symbol demapping, turbo decoding, and transport block reassembling. An LTE symbol is received from the environment each 71428 ns and OFDM demodulation is performed for each received LTE symbol. Pilot symbols are inserted in the received data frames to facilitate channel estimation. The effects of channel propagation are compensated through the equalization function. Process *Symbol Demapper* represents the interface between symbol level processing and bit processing. Channel decoding is performed through a turbo decoder algorithm. Process *Transport Block Reassembly* receives a binary data block through relation *Segmented Block*. Data blocks are then transmitted to the medium access control layer each 1 ms, when 14 OFDM symbols have been received and processed. Different configurations are supported by the receiver according to the parameters of the received data frames. The parameters concern the size of the LTE symbol, the modulation scheme, the number of iterations of the channel decoder, and the number of data blocks allocated to each user. Table 2 gives the main possible values of data frame parameters. These parameters directly influence the computation and communication workloads of the application.

The studied platform is made of two computation resources, one communication bus, and four communication interfaces. The turbo decoder process is implemented as a dedicated resource whereas other functions are allocated to a digital signal processor. Functions allocated to processor P1 are thus sequentially executed. The communication of relation *Code block* is managed by interfaces IF1 and IF2. The communication of relation *Segmented block* is managed by interfaces IF3 and IF4. As previously, we assumed that the interfaces cannot manage simultaneously data transfer and read/write operations.

Table 2 Parameters of LTE data frames.

Modulation scheme	QPSK, 16QAM, 64QAM
Number of active sub-carriers	72, 180, 300, 600, 900, 1200
Number of data blocks	6, 15, 25, 50, 75, 100
Signal bandwith (MHz)	1.4, 3, 5, 10, 15, 20

We captured this system model using the Intel CoFluent Studio framework to evaluate the usage of the computation and communication resources and to estimate the required computational complexity. A workload model of the application was built to capture functions behavior and data dependencies. We evaluated the number of operations required for each function [3]. The computational complexity of each function depends on the parameters of the received data frames. The system model of Fig. 21 was first captured through three views and it was simulated using the trace-driven approach. The proposed simulation approach was then applied to simulate the architecture model. We used the presented Petri net patterns to formulate the dependencies among the application processes. We also considered the limited concurrency of communication interfaces. A temporal dependency graph was created to compute the synchronization instants during simulation. The size of related array G was 84. The simulated model had the same organization as in Fig. 14. Part (a) of Fig. 22 illustrates achieved evolution over the simulation time of functions allocated to P1[6]. Using the proposed simulation approach, intermediate synchronization instants among functions were locally computed for each LTE symbol. In the considered case study, a Transport block was not produced each time a LTE symbol was received and the intermediate synchronization instants were thus successively computed and stored before a synchronization instant at the output of the executable model was computed. Part (b) of Fig. 22 illustrates the computational complexity over the observation time. The observed results correspond to the reception of a LTE frame with the following parameters: modulation scheme: QPSK, number of active sub-carriers: 300, number of data blocks: 12. A maximum computational complexity per time unit of 1.6 GOPS was observed for P1 and 221.7 GOPS was observed for P2. Application of the proposed simulation approach provided same timing accuracy compared to trace-driven simulation. We evaluated the achieved simulation speed-up with the same conditions as in the previous experiments. A mean

[6] Delays due to communications between functions are not detailed for clarity reasons.

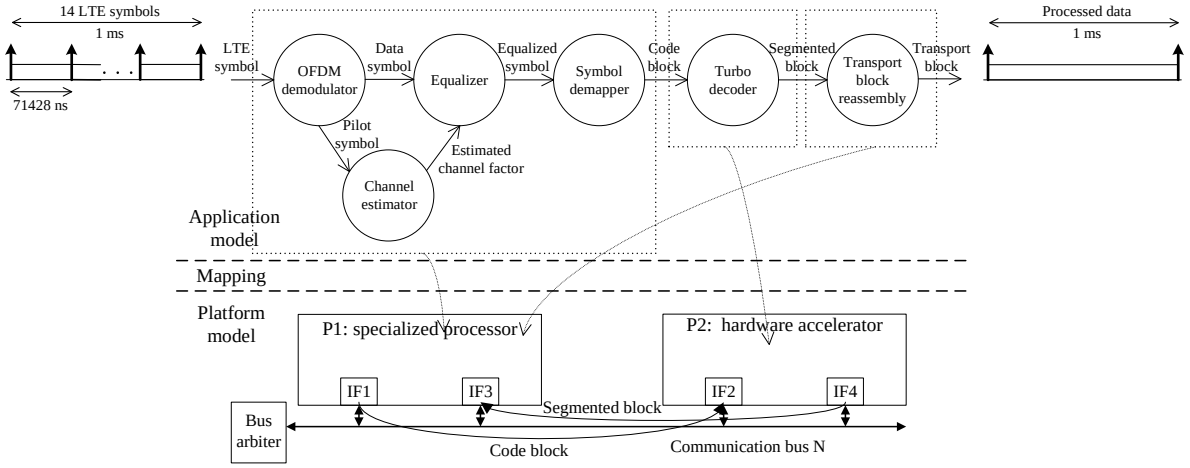


Fig. 21 Evaluated LTE communication receiver.

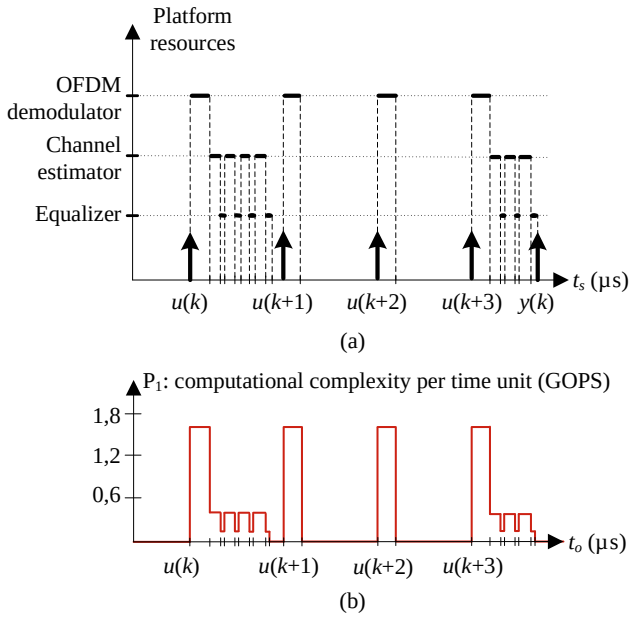


Fig. 22 Evolution of the resource usage of the studied system over the simulation time (a) and the observation time (b).

simulation speed-up by a factor of 4 was achieved with no loss of timing accuracy. Other organizations with the same application and different mappings could be considered. Using the proposed approach, graphs with similar complexity could thus be created and similar simulation speed-up could be expected.

In this paper, we have presented application of the simulation approach for contention at communication resources. Other forms of resource contention could be taken into account. Petri nets are well adapted to express other forms of resource access policies. In the case of computation resources, Petri nets could be built to describe execution of multiple functions sharing the same processor. In the proposed approach, instants at which resources are used would still be locally determined. In our future work, application of the approach to other scheduling and management policies will be considered. Besides, according to the application and platform models, the size of the Petri net, and the related temporal dependency graph, can significantly increase. One important limitation of the approach is thus caused by the size of the required Petri net to appropriately describe resource management.

In this paper, the models supporting the proposed simulation approach were created manually. However, creation of temporal dependency graphs could be done in a systematic way by analyzing the system model. This requires full validation of patterns related to modeling statements. Timed Petri net and its implementation through temporal dependency graph represent convenient formalisms to define and implement patterns. A generation tool could thus assist automatic generation of simulation models and significantly reduce the modeling effort. Generation of temporal dependency graphs would take place during the generation phase of executable simulation models. This tool development is a part of our future work. The generation process should also concern the creation of an executable model for the equivalent model presented in Fig. 14. In the future, this equivalent model should be extended to appropriately handle input asynchronous events caused by interrupts.

7 Conclusion

In this article, we presented a hybrid simulation approach that improves the simulation efficiency of performance models of hardware-software architectures. In the scope of this article, we focused on data flow oriented systems with possible contentions at shared communication resources. We have shown that the proposed simulation techniques provide significant simulation speed-up with no loss of timing accuracy.

Existing reduction techniques of timed Petri net could be used for reducing the state-explosion problem of Petri nets. These techniques aim at reducing the size of a Petri net while retaining properties. Such techniques would allow the size of the manipulated graphs to be decreased and application of the simulation approach to more complex systems. Future work will address the development of a generation tool to support the creation process of simulation models. Advanced modeling statements and arbitration policies for multi-processor platforms will also be addressed.

References

1. Arpinen, T., Salminen, E., Hämäläinen, T.D., Hännikäinen, M.: Performance evaluation of UML-2 modeled embedded streaming applications with system-level simulation. EURASIP Journal on Embedded Systems **2009**, 826296 (2009)
2. Baccelli, F., Cohen, G., Olsder, G., Quadrat, J.: Synchronization and linearity, an algebra for discrete event systems. Wiley & Sons Ltd, New York (1992)
3. Berkmann, J., Carbonelli, C., Dietrich, F., Drewes, C., Xu, W.: On 3G LTE terminal implementation - standard, algorithms, complexities and challenges. In: 2008 International Wireless Communications and Mobile Computing Conference, pp. 970–975 (2008). DOI 10.1109/IWCMC.2008.168
4. Bobrek, A., Pieper, J.J., Nelson, J.E., Paul, J., Thomas, D.E.: Modeling shared resource contention using a hybrid simulation/analytical approach. In: Proc. Design, Automation and Test in Europe (DATE'04), pp. 1144–1149. Paris, France (2004)
5. Cai, L., Gajski, D.: Transaction level modeling: an overview. In: In Proc. of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'03), pp. pp. 19–24. Newport Beach, CA, USA (2003)
6. Chakraborty, S., Künzli, S., Thiele, L.: A general framework for analyzing system properties in platform-based embedded system designs. In: Proc. Design, Automation and Test in Europe (DATE'03), pp. 190–195. Munich, Germany (2003)
7. Chen, S.Y., Chen, C.H., Tsay, R.S.: An activity-sensitive contention delay model for highly efficient deterministic full-system simulations. In: In Proc. of Design, Automation and Test in Europe (DATE'14) (2014)
8. Dahlman, E., Parkvall, S., Skold, J., Beming, P.: 3G evolution: HSPA and LTE for mobile standard. Elsevier Academic Press, Amsterdam (2008)
9. Densmore, D., Passerone, R., Sangiovanni-Vincentelli, A.: A platform-based taxonomy for ESL design. IEEE Design and Test of Computers pp. 359–374 (2006)
10. Dömer, R., Gerstlauer, A., Peng, J., Shin, D., Cai, L., Yu, H., Gajski, D.: System-on-chip environment: A SpecC-based framework for heterogeneous MPSOC design. EURASIP J. Embed. Syst. **2008**(3), pp.1–13 (2008)
11. Erbas, C., Pimentel, A.D., Thompson, M., Polstra, S.: A framework for system-level modeling and simulation of embedded systems architectures. EURASIP J. Embed. Syst. **2007**, pp. 1–11 (2007)
12. Gajski, D.D., Abid, S., Gerstlauer, A., Schirner, G.: Embedded system design: modeling, synthesis and verification. Springer (2009)
13. Gerstlauer, A., Haubelt, C., Pimentel, A.D., Stefanov, T., Gajski, D.D., Teich, J.: Electronic system-level synthesis methodologies. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **28**(10), 1517–1530 (2009)
14. IEEE computer society: IEEE standard SystemC language reference manual. IEEE Std. 1666–2011 (2011). URL http://standards.ieee.org/getieee/1666/
15. Intel: Intel cofluent studio (2017). URL http://www.intel.com/content/www/us/en/cofluent/intel-cofluent-studio.html
16. Jalier, C., Lattard, D., Jerraya, A.A., Sassatelli, G., Benoit, P., Torres, L.: Heterogeneous vs homogeneous MPSoC approaches for a mobile LTE modem. In: Design, Automation Test in Europe Conference Exhibition (DATE 2010), pp. 184–189 (2010). DOI 10.1109/DATE.2010.5457213
17. Keinert, J., Streubühr, M., Schlichter, T., Falk, J., Gladigau, J., Haubelt, C., Teich, J., Meredith, M.: SystemCoDesigner-an automatic ESL synthesis approach by design space exploration and behavior synthesis for streaming applications. ACM Trans. Des. Autom. Electro. Syst. **14**(1), pp.1–23 (2009)
18. Khaligh, R.S., Radetzki, M.: Modeling constructs and kernel for parallel simulation of accuracy adaptive tlms. In: 2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010), pp. 1183–1188 (2010). DOI 10.1109/DATE.2010.5456987
19. Kreku, J., Hoppari, M., Kestilä, T., Qu, Y., Soininen, J., Andersson, P., Tiensyrjä, K.: Combining UML2 application and SystemC platform modelling for performance evaluation of real-time embedded systems. EURASIP Journal on Embedded Systems **2008** (2008)
20. Künzli, S., Poletti, F., Benini, L., Thiele, L.: Combining simulation and formal methods for system-level performance analysis. In: Design, Automation and Test in Europe (DATE), pp. 236–241. Munich, Germany (2006)
21. Le Nours, S.: Timing correction technique for fast and accurate state-based performance models. In: Proc. Forum on specification and design languages (FDL'15) (2015)
22. Le Nours, S., Postula, A., Bergmann, N.: A dynamic computation method for fast and accurate performance evaluation of multi-core architectures. In: Proc. Design, Automation, and Test in Europe (DATE'14). Dresden, Germany (2014)
23. Lieverse, P., van der Wolf, P., Vissers, K., Deprettere, E.: A methodology for architecture exploration of heterogeneous signal processing systems. Journal of VLSI Signal Processing **29**, 197–207 (2001)
24. Lu, K., Muller-Gritschneder, D., Schlichtmann, U.: Analytical timing estimation for temporally decoupled TLMs considering resource conflicts. In: Proc. Design, Automation and Test in Europe (DATE'13), pp. 1161–1166. Grenoble, France (2013)
25. Mirabilis: Mirabilis visualsim. URL http://www.mirabilisdesign.com
26. Pimentel, A.D., Thompson, M., Polstra, S., Erbas, C.: Calibration of abstract performance models for system-level design space exploration. Journal of Signal Processing Systems **50**, pp. 99–114 (2008)
27. Razaghi, P., Gerstlauer, A.: Host-compiled multicore system simulation for early real-time performance evaluation. ACM Trans. Embed. Comput. Syst. **13**(5s), 166:1–166:26 (2014). DOI 10.1145/2678020. URL http://doi.acm.org/10.1145/2678020
28. Savoiu, N., Shukla, S., Gupta, R.: Improving SystemC simulation through petri net reductions. In: Proceedings of International Conference on Formal Methods and Models for Co-Design (MEMOCODE'05), pp. 131–140 (2005)
29. Schirner, G., Dömer, R.: Result-oriented modeling - a novel technique for fast and accurate TLM. IEEE Transactions on computer-aided design of integrated circuits and systems **26**(9), 1688–1699 (2007)
30. Schirner, G., Dömer, R.: Introducing preemptive scheduling in abstract RTOS models using result oriented modeling. In: Proc. Design, Automation and Test in Europe (DATE'08), pp. 122–127. Munich, Germany (2008)

31. Schirner, G., Dömer, R.: Quantitative analysis of the speed/accuracy trade-off in transaction level modeling. ACM Trans. Embed. Comput. Syst. **8**(1), 4:1–4:29 (2009). DOI 10.1145/1457246.1457250. URL http://doi.acm.org/10.1145/1457246.1457250
32. Seidner, C., Roux, O.H.: Formal methods for systems engineering behavior models. IEEE Transactions on industrial informatics **4**, 280–291 (2008)
33. SpaceCodesign: Space studio. URL http://www.spacecodesign.com
34. Stattelmann, S., Bringmann, O., Rosenstiel, W.: Fast and accurate resource conflict simulation for performance analysis of multi-core systems. In: Proc. Design, Automation and Test in Europe (DATE'11). Grenoble France (2011)