Stopwatch Automata-Based Model for Efficient Schedulability Analysis of Modular Computer Systems*

Alevtina Glonina and Anatoly Bahmurov

Lomonosov Moscow State University, Moscow, Russia
{alevtina, bahmurov}@lvk.cs.msu.su

Abstract. In this paper we propose a stopwatch automata-based model of a modular computer system operation. This model provides an ability to perform schedulability analysis for a wide class of modular computer systems. It is formally proven that the model satisfies a set of correctness requirements. It is also proven that all the traces, generated by the model interpretation, are equivalent for schedulability analysis purposes. The traces equivalence allows to use any trace for analysis and therefore the proposed approach is much more efficient than Model Checking, especially for parallel systems with many simultaneous events. The software implementation of the proposed approach is also presented in the paper.

Keywords: stopwatch automata, integrated modular avionics, simulation, schedulability analysis

1 Introduction

Nowadays modular approach to computer systems design is replacing the old federated approach. We consider Integrated Modular Avionics (IMA) [1] systems as an example of modular computer systems, but the proposed approach can be also applied for other modular architectures (e.g. [2] and [3]).

An IMA system consists of standardized hardware modules containing multicore processors connected by a switched network with virtual links. There can be several module types in a system with different processors performance.

A module hardware resources are shared by several applications, called partitions. Every partition is mapped to one of the processing cores. One core can be shared by several partitions. A partition has its own memory space and execution time slots, called windows. A core's scheduling period is divided into windows and each window corresponds to one of the core's partitions.

A partition contains a set of tasks. A task is characterized by priority, period, deadline and worst case execution time (WCET) on every processor type. Every task period an instance of the task (called a job) must be executed. There can be data dependencies between tasks with the same period: current job of receiver

* The work is supported by the RFBR grant 17-07-01566

task can't be executed until it receives data from corresponding jobs of all senders tasks.

Every partition has its own task scheduler which controls tasks execution. Schedulers usually work according to dynamic algorithms. The most common algorithm is fixed-priority preemptive scheduling (FPPS) algorithm. Every job must complete within its deadline. If a job's deadline is reached this job can not be executed anymore.

System configuration contains characteristics of hardware modules and partitions, mapping partitions to cores and windows sets for cores. The configuration is called schedulable if all the jobs complete within their deadlines. During system design multiple potential configurations are considered and for each of them schedulability analysis must be performed.

There are many schedulability analysis approaches, but some of them do not consider all modular systems features (e.g. [4]) and others have too high computational complexity (e.g Model Checking [5]). Another approach is generating system operation trace and then analyzing this trace. Unfortunately, all the existing tools for it have essential drawbacks: some also do not consider all modular systems features (e.g [6]), others support only manual model development (e.g [7]) and almost all do not support any formal proving of model correctness (e.g. [8]). In this work we propose a general model for modular system operation, which can be used for required trace generation and overcomes these drawbacks.

As a modular system consists of standardized components, our model also consists of standardized sub-models. The key idea is to model every component type with a parametric stopwatch automaton with specified interface. The whole model is a parametric Network of Stopwatch Automata (NSA) [9]. System model for a given configuration can be constructed automatically. The formalism of NSA allowed us to prove formally that our model satisfies correctness requirements necessary for using it for schedulability analysis. We also proved that for a given configuration all interpretations of the proposed model are equivalent. This fact allows to use any single model interpretation for schedulability analysis in contrast to Model Checking where all possible interpretations are considered.

The rest of the paper is structured as follows: in Sect.2 necessary formal definitions are given and our model is presented, in Sect.3 the model determinism and correctness are proven and in Sect.4 the software implementation of the proposed approach is described and experimental results are discussed.

2 The Model of Modular System Operation

2.1 Formal Definitions

A system configuration is a tuple $\langle HW, WL, Bind, Sched \rangle$, where

- $HW = \{HW_i\}_{i=1}^{N}$ — processing cores; $Type : HW \to \overline{1, N_t}$ — core type ($N_t \in \mathbb{N}$ — number of core types); $Mod : HW \to \overline{1, N_m}$ — module number for a core ($N_m \in \mathbb{N}$ — number of modules);
- $WL =< Part, G >$ — workload, where:

- $Part = \{Part_i = \langle T_i, A_i \rangle\}_{i=1}^{M}$ — partitions, where:
  - $T_i = \{T_{ij}\}_{j=1}^{K_i}$ — tasks, each characterized by priority (pr_{ij}), WCETs on different core types ($\overline{C_{ij}} = (C_{ij}^1, ..., C_{ij}^{N_t})$), period ($P_{ij}$), deadline ($D_{ij}$);
  - A_i — scheduling algorithm type;
- $G = \langle \cup_{i=1}^{M} T_i, \{Msg_j\}_{j=1}^{H} \rangle$ — data flow graph, where Msg_j corresponds to a message and is characterized by sender and receiver tasks and maximum durations of transfer through memory and through network;

– $Bind : Part \to HW$ — partitions binding to cores;
– $Sched = \{\{\langle Start_{ij}, End_{ij} \rangle\}_{j=1}^{N_i^w}\}_{i=1}^{M}$ — partitions schedule, which is repeated periodically with a period L equal to the least common multiple of all the tasks periods; $N_i^w \in \mathbb{N}$ — number of windows for the i-th partition; $Start_{ij}$, $End_{ij} \in \overline{0, L}$ — start time and end time for j-th window of i-th partition.

Let $CONF$ be a set of all possible system configurations.

For a task T_{ij} a set of jobs $W_{ij} = \{w_{ijk}\}_{k=1}^{L/P_{ij}}$ is defined.

Let $e = \langle Type, Src, t \rangle$ be an event, where $Type \in \{EX, PR, FIN\}$ corresponds to start or continuation of a job execution (EX), job preemption (PR) and finish of a job execution due to its completion or reaching deadline (FIN); $Src \in W_{ij}$ is a source job for the event; $t \in \overline{1, L}$ is a timestamp. Let E be a set of possible events.

A system operation trace is a set of events, therefore it is a subset of E. Let $TR \in 2^E$ be a set of all possible traces.

Design problems for IMA systems are commonly being solved under the following assumptions (e.g. in [8, 10], considering industrial avionics systems):

– Every job's execution time is equal to its WCET.
– Every message transfer delay is also equal to its worst case; typical avionics networks (e.g. AFDX) allow to obtain safe estimations for these delays.
– Scheduling algorithms are deterministic (e.g. in ARINC 653 systems [1]).

Under these assumptions system operation is deterministic and corresponds to a worst-case scenario, i.e. only one trace corresponds to a configuration and this trace can be used for schedulability analysis. Therefore the mapping $Q : CONF \to TR$ exists and $\forall conf \in CONF : \exists! Q(conf)$.

Let R_{ijk} be a number of executing intervals for a job w_{ijk}. Then an ordered subtrace for this job is:

– empty, if $R_{ijk} = 0$;
– $\langle EX, w_{ijk}, t_0 \rangle, \langle FIN, w_{ijk}, t_1 \rangle$, if $R_{ijk} = 1$;
– $\langle EX, w_{ijk}, t_0 \rangle, \langle PR, w_{ijk}, t_1 \rangle, ..., \langle EX, w_{ijk}, t_{2R_{ijk}-2} \rangle, \langle FIN, w_{ijk}, t_{2R_{ijk}-1} \rangle$, if $R_{ijk} > 1$.

In these terms the schedulability criterion has the following form:
$\forall w_{ijk}, i \in \overline{1, M}, j \in \overline{1, K_i}, k \in \overline{1, L/P_{ij}} : \sum_{r=1}^{R_{ijk}} (t_{2r-1} - t_{2r-2}) = C_{ij}^{Type(Bind(Part_i))}$

In this paper we consider the problem of building the system operation model, the interpretation of which defines the mapping Q. This model is necessary for checking the schedulability criterion for a given configuration.

2.2 Networks of Stopwatch Automata

First of all the mathematical formalism for system operation description must be chosen. It should meet the following requirements:

- ability to model such aspects of system operation as queues, preemption, parallel functioning of different schedulers;
- ability to obtain a time trace of model interpretation;
- ability to formalize and check requirements to models;
- existence of software tools for modeling and verification.

We reviewed several formalisms found in the literature and the formalism of stopwatch automata networks [9, 11] was chosen, as it meets all the requirements and has the best program support of modeling and verification. Now we give a brief description of the formalism.

A stopwatch automaton is a finite automaton (denoted graphically by a graph containing a set of nodes or locations and a set of labeled edges) extended with integer variables and clocks. Each variable has a bounded domain and an initial value. A clock is a special real-valued variable, that can be compared with integer variables or with other clocks, reset to zero, stopped and later resumed with the same value. All the clocks are initialized with zero and then increase synchronously (except for the stopped clocks) with the same rate.

An edge represents an action transition and has three labels: a guard label, a synchronization (will be explained later) label and an update label. A transition can be taken when clocks and variables satisfy the guard and synchronization can be performed. During action transitions, synchronizations and updates of clocks and variables are performed. A location has a label called an invariant, which is a predicate over variables and clocks. An automaton may remain in a location as long as the invariant of the location is true. For an automaton an initial location is defined.

In addition to action transitions, represented by automaton edges, there are delay transitions, corresponding to synchronous clock increasing by the same real value. All the clocks (except for the stopped clocks) can be increased by a value of d if their values increased by d satisfy current location invariant. Some locations can be labeled as *committed*. No delay transitions can be performed if an automaton current location is committed.

A network of stopwatch automata (NSA) is a set of several automata, operating synchronously. Communications between the automata are performed by using shared variables and channels.

A channel is a mechanism for automata synchronous communication. Every automaton edge has a synchronization label, which can be either an empty label (for internal transitions) or a synchronization action. There are two complementary types of such actions: *sending* and *receiving* a signal through a channel. And there are two types of channels: binary and broadcast. Two transitions in different automata can synchronize via a binary channel if the guards of both transitions are satisfied, and they have complementary synchronization actions. A transition with binary synchronization action can be performed if and only

if the transition in the other automaton with complementary action can be performed. When synchronization is performed the current locations of both automata are changed, i.e. the both transitions are performed simultaneously. $N + 1$ automata can synchronize via a broadcast channel if the transition with sending action is enabled and N transitions with receiving action are enabled.

More formally a stopwatch automaton is a tuple $\langle L, l_0, U, C, V, \overline{v_0}, AU, AS, E, I, P\rangle$, where

- $L, l_0 \in L, U \subseteq L$ — finite set of locations, initial location and set of committed locations;
- C — set of clocks;
- $V, \overline{v_0}$ — set of integer variables and their initial values;
- AU, AS — sets of updating and synchronization actions;
- E — set of edges, $E \subseteq L \times B(C, V) \times AU \times AS \times L$, where $B(C, V)$ is a set of predicates over C and V
- $I : L \to B(C, V)$ associates invariants to locations;
- $P : L \times C \to B(\emptyset, V)$ associates progress conditions to locations and clocks;

Let $A = A_1|...|A_n$ be an NSA, where $A_i = \langle L_i, l_i^0, U_i, C_i, V_i, \overline{v_i^0}, AU_i, AS_i, E_i, I_i, P_i\rangle$. A state of the NSA is a tuple $\langle \overline{l}, \overline{c}, \overline{v}\rangle \in (L_1 \times ... \times L_n) \times \mathbb{R}_{\geq 0}^{|C|} \times \mathbb{Z}^{|V|}$, where $V = \cup V_i$, $C = \cup C_i$. A sequence (may be infinite) of action and delay transitions between states $\langle \overline{l_0}, \overline{c_0}, \overline{v_0}\rangle \to \langle \overline{l_1}, \overline{c_1}, \overline{v_1}\rangle \to ... \to \langle \overline{l_i}, \overline{c_i}, \overline{v_i}\rangle \to ...$ is a run of an NSA. An NSA usually has many (may be infinitely) possible runs.

2.3 General Model of Modular System Operation

To present our model we have to introduce several definitions.

A parametric stopwatch automata, or *concrete automata type*, is a tuple $\langle L, l_0, U, C, V, \overline{p}, AU, AS, E, I, P\rangle$, where $\overline{p}$ is a vector of unknown integer-valued parameters. An automaton's shared variables and possible synchronization actions comprise the *automaton interface*. *Base automata type* is a pair of sets $\langle V_b, AS_b\rangle$, where V_b is a set of shared variables and AS_b is a set of synchronization actions. A concrete automata type *implements* a base automata type if $V_b \subseteq V$, $AS_b \subseteq AS$.

A set of base automata types is a *general NSA*. A set of concrete automata types is a *concrete NSA*. A concrete NSA *implements* a general NSA if each base automata type in the general NSA is implemented by one or more concrete automata type in the concrete NSA and logic relations between concrete automata types corresponds to relations (i.e. rules defining, which implementations of base automata types must communicate) between base automata types.

Model time is a value of a special clock, which is never stopped or reset. *Synchronization event* is a tuple $\langle CH, A, t\rangle$, where CH is the channel, A is a set of automata instances, participating in the synchronization, t is the model time of synchronization. *NSA trace* is a set of synchronization events, generated by the network.

We propose to represent the general model of modular system operation as a general NSA.

The following shared variables and channels are used for automata communication in the proposed model:

- variables $\texttt{is\_ready}_{ij}$, $\texttt{is\_failed}_{ij}$, $\texttt{prio}_{ij}$, $\texttt{deadline}_{ij}$ $i \in \overline{1, M}, j \in \overline{1, K_i}$, each corresponding to a job readiness, reaching its deadline and a task characteristics;
- variables $\texttt{is\_data\_ready}_h$, $h \in \overline{1, H}$, each corresponding to a message delivery through the hth virtual link;
- channels $\texttt{wakeup}_i$, $\texttt{sleep}_i$, $\texttt{ready}_i$, $\texttt{finished}_i$ $i \in \overline{1, M}$, each corresponding to a window start and finish, a ready job arrival and its finish; a job finishes either due to its completion or due to reaching its deadline;
- channels $\texttt{exec}_{ij}$, $\texttt{preempt}_{ij}$, $i \in \overline{1, M}$, $j \in \overline{1, K_i}$, each corresponding to a job execution start (or resumption) and preemption;
- broadcast channels $\texttt{send}_{ij}$, $\texttt{receive}_{ij}$, $i \in \overline{1, M}$, $j \in \overline{1, K_i}$, each corresponding to receiving data from a sender job and sending data to all receiver jobs.

The general NSA consists of the following base automata types:

1. `T` base automata type modeling a task. As a task deadline is less or equal to its period, there can be only one active job of a task at a given moment. `T` is defined by following interface:

- receiving signals through channels `exec` and `preempt`;
- sending signals through channels `ready`, `finished`, `send`, `receive`;
- changing variables `is_ready`, `is_failed`, $\texttt{is\_data\_ready}_h$;

2. `TS` base automata type modeling a task scheduler for a partition. It is defined by following interface:

- receiving signals through channels `wakeup`, `sleep`, `ready`, `finished`;
- sending signals through channels $\texttt{exec}_j$, $\texttt{preempt}_j$; the j-th channel corresponds to the j-th task of the partition;
- reading variables $\texttt{is\_ready}_j$, $\texttt{prio}_j$, $\texttt{deadline}_j$; the j-th variable corresponds to the j-th task of the partition.

3. `CS` base automata type modeling a core scheduler (scheduling partitions for a core). It is defined by the following interface:

- sending signals through channels $\texttt{wakeup}_i$ and $\texttt{sleep}_i$; the i-th channel corresponds to the ith partition.

4. `L` base automata type modeling a virtual link. It is defined by the following interface:

- receiving signals through a broadcast channel `send`;
- sending signals through a broadcast channel `receive`;
- changing variable `is_data_ready`.

The structure of the proposed general model of modular system operation is shown on Fig. 1.

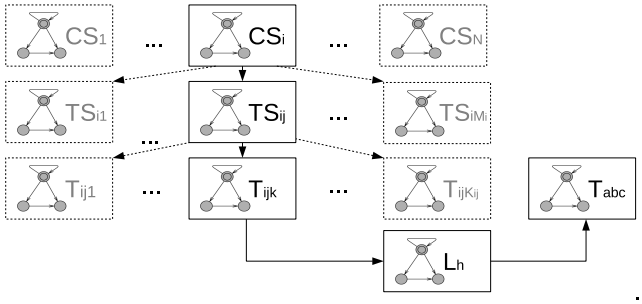

Fig. 1. The structure of the general NSA type modeling modular system operation.

A concrete NSA implementing the proposed general NSA is a parametric model of modular system operation. Our concrete NSA has the following concrete automata types, implementing base automata types: task model, core scheduler model, virtual link model, FPPS scheduler, FPNPS scheduler and EDF scheduler. For a given concrete NSA and a system configuration an NSA instance can be constructed by the Algorithm 1.

Algorithm 1: An NSA instance construction

Data: $conf \in CONF$, concrete NSA
Result: NSA instance modeling system of $conf$ configuration
begin
  for $i \in \overline{1, N}$ **do**
    for $j \in \overline{1, M} : Bind(Part_j) = HW_i$ **do**
      create channels ready_j, $\mathsf{finished}_j$, wakeup_j, sleep_j;
      for $k \in \overline{1, K_i}$ **do**
        create channels exec_{jk}, $\mathsf{preempt}_{jk}$, send_{jk}, $\mathsf{receive}_{jk}$ and variables $\mathsf{is\_ready}_{jk}$, prio_{jk}, $\mathsf{deadline}_{jk}$, $\mathsf{is\_data\_ready}_h$ (each corresponding to a virtual link, where jkth task is a receiver);
        create an automaton implementing T, initialize its interface with channels exec_{jk}, $\mathsf{preempt}_{jk}$, send_{jk}, $\mathsf{receive}_{jk}$, ready_j, $\mathsf{finished}_j$ and variables $\mathsf{is\_ready}_{jk}$, prio_{jk}, $\mathsf{deadline}_{jk}$, $\mathsf{is\_data\_ready}_h$;
      create an automaton implementing TS and corresponding to A_j for jth partition, initialize its interface with channels exec_{jk}, $\mathsf{preempt}_{jk}$, ready_j, $\mathsf{finished}_j$, wakeup_j, sleep_j and variables $\mathsf{is\_ready}_{jk}$, prio_{jk}, $\mathsf{deadline}_{jk}$ $(k \in \overline{1, K_i})$;
    create an automaton implementing CS for ith core and initialize its interface with corresponding channels wakeup_j, sleep_j;
  for $h \in \overline{1, H}$ **do**
    create an automaton implementing L, initialize its interface with corresponding channels $\mathsf{send}_{j_1k_1}$, $\mathsf{receive}_{j_2k_2}$ and variable $\mathsf{is\_data\_ready}_h$.

By construction there is an automaton of appropriate type for every system component and automata interfaces for logical connections between components. Automata parameters correspond to a system configuration parameters. Therefore there is unambiguous correspondence between a system configuration and a model instance.

A system operation trace, which is necessary for checking the schedulability criterion, can be unambiguously obtained from the corresponding model trace (i.e. a trace of the NSA instance).

3 Correctness and Determinism

Modular systems specifications contain correctness requirements to system components operation and to the whole system operation. These requirements specify correct events sequences and delays between events of given types. In order to ensure schedulability analysis correctness, our model must satisfy correctness requirements, which are applicable at the chosen abstraction level.

We call a model deterministic if a trace generated by its run is uniquely determined. This determinism is crucial for schedulability analysis of large systems with many simultaneous events, because it allows to use any of the NSA runs for a trace generation in contrast to model-checking where all possible runs are to be considered.

Correctness requirements to system components models (i.e. parametric automata) can be checked automatically by a verifier. For this purpose we chose "observers" approach [12], which is successfully used in practice.

One observer automaton usually corresponds to one requirement. The observer is an automaton, which operates synchronously with a given automaton and does not block any synchronization. The observer has one "bad" location and all incorrect synchronization event sequences or incorrect delays lead the observer to the "bad" location. The reachability of the "bad" location means that an incorrect event sequences can be generated by the given automaton and therefore it does not satisfy the requirement. As the given automaton is parametric and must operate correctly with all possible parameters values, its observer non-deterministically sets each parameter to one of possible values.

We derive correctness requirements to system components from system specifications, construct an observer for each requirement and automatically check with UPPAAL [11] verifier that "bad" locations are unreachable. Such proof was performed for a set of requirements derived from ARINC 653 specification [1] and the set of concrete automata types described in Sect.2.3.

Let us consider a correctness requirement example and build its observer:

For every partition at any time zero or one job can be executed.

This is the requirement to `TS` base automata type and all the `TS` implementations must satisfy this requirement. In terms of synchronization events, a job of task T_{jk} is executed between synchronizations through channels $\texttt{exec}_{jk}$ and $\texttt{preempt}_{jk}$, and through channels $\texttt{exec}_{jk}$ and $\texttt{finished}_{j}$. It means that any

synchronization through $\texttt{exec}_{jk}$ must be followed by a synchronization through $\texttt{preempt}_{jk}$ or $\texttt{finished}_j$. The corresponding observer is shown on Fig. 2.

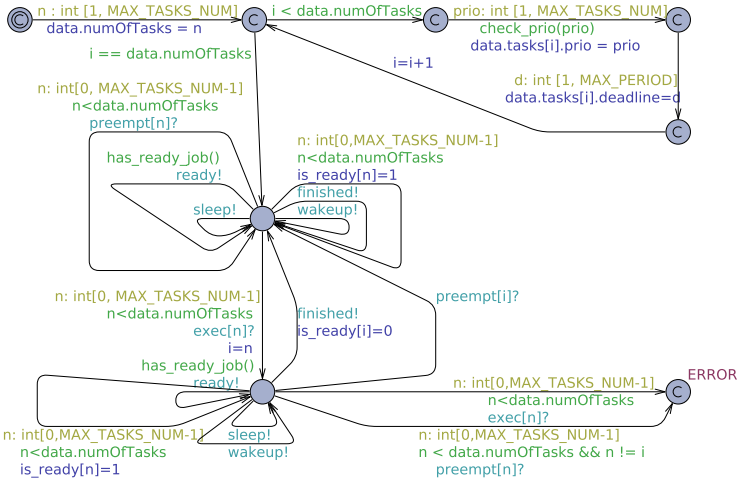

Fig. 2. The observer automaton for the requirement to TS automata

Satisfaction of the requirements to the whole general model can't be proven automatically because the number of automata of different types in the model is unknown in general. Thus, we have to prove the satisfaction of these requirements manually. This proof implies that all the models instances constructed by the algorithm 1 satisfy these requirements. Our proof is based on the satisfaction of the requirements to components models which are proven automatically.

This is an example of a requirement to the whole model and its proof:

If one task depends on another, then start time for any job of the receiver task is more or equal the completion time for the corresponding job of the sender task plus the upper bound of the message transfer delay

The satisfaction of the following requirements to components models were proven automatically:

1. Every job sends data to its output virtual links after its completion.

2. A message transfer delay trough a virtual link is equal to its pessimistic upper bound.

3. A job of receiver task can't be executed until it receives data from corresponding jobs of all senders tasks.

The satisfaction of these requirements implies the satisfaction of the given requirement to the whole model.

The model determinism proof is based on the previously proven satisfaction of the correctness requirements.

Suppose by contradiction that two different NSA traces can be generated by the model interpretation for a given configuration. Let the both traces be

partially ordered by events time. Thus, a set of events is bound to every time point in every trace. Let t_i be the first time point, which has different events sets for given traces. It means that at least one event is contained in one event set and is absent in the other. Suppose that this event is a synchronization through $\texttt{finished}_j$. As all previous events sets are equal for the traces, there are two alternatives:

1. Some job executes on the processor for WCET time units according to the first trace (where the event is contained). But it means that this job's cumulative time of execution on the processor is more than WCET according to the second trace (where the event is absent).

2. Some job reaches its deadline according to the first trace. But it means that this job is not removed from the processor after its deadline is reached according to the second trace.

Both the alternatives are impossible, because they imply violation of the requirements, which satisfaction was previously proven. Therefore the supposition is impossible. For other events types the proof is similar.

So we proved that the proposed model satisfies correctness requirements and all the traces generated by its interpretation are equal. It was also shown that there is unambiguous correspondence between a system configuration and a generated model instance and between a system trace and a model interpretation trace. Therefore schedulability analysis (checking the criterion specified in Sect.2.1) performed by using this model is correct.

4 Implementation and Experiments

In order to test the applicability of the proposed approach in practice we implemented it in software. The concrete automata types modeling concrete types of system components were developed and verified using UPPAAL [11] toolset. These concrete automata types are contained in an automata components models library. A user can develop, verify and add to the library own models. As UPPAAL doesn't have commandline interface for NSA interpretation, we developed our own NSA simulation library in C++ and a translator from UPPAAL to C++ automata representation. The library of automata models for components was translated to a library of software models. Models from the library compose the parametric software model of system operation (see Fig. 3).

We compared the proposed approach with Model Checking using the same NSA. The results of the experiments confirm that our approach is much more efficient (see Table 1).

Table 1. Execution times for various number of jobs

Number of jobs	10	11	12	13	14	15	16	17	18
Model Checking (seconds)	0.57	1.16	2.22	5.05	10.43	23.51	48.13	112.28	215.91
Proposed Approach (seconds)	0.027	0.027	0.028	0.030	0.031	0.032	0.033	0.035	0.036

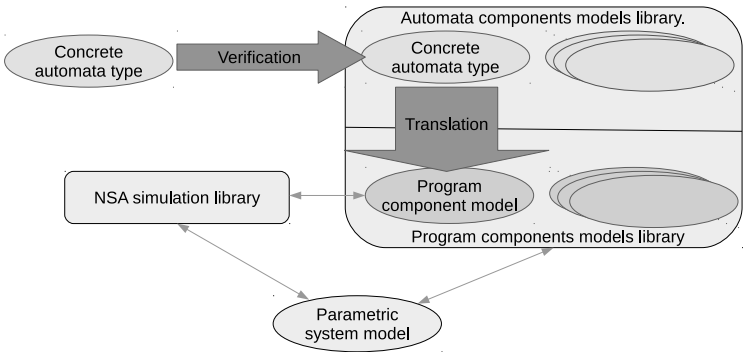

Fig. 3. The scheme of the parametric modular system operation model organization.

We also integrated the parametric model with an IMA scheduling tool, which searches the optimum IMA configuration among possible configurations [8]. On every iteration the scheduling algorithm chooses a configuration to be checked for schedulability. Then an XML file with the configuration description is generated and passed to the parametric model. After that a model instance is created and run and it trace is passed back to the scheduling tool, which performs schedulability analysis. Unschedulable configurations are discarded by the scheduling algorithm and schedulable ones are considered as candidate solutions. The experiments showed that a model instance construction and interpretation take about several seconds for configurations of same complexity as configurations of industrial avionics systems (about 11 seconds for a configuration with 12500 jobs). Thus it was shown that our approach is applicable in practice.

5 Conclusion

We developed a general model of a modular computer system operation based on the NSA formalism. The model can be used for schedulability analysis of such systems configurations. It was proven that our model is deterministic and correct, and therefore the analysis is performed correctly. The model determinism (in terms of jobs start, finish and preemption) makes our approach is significantly more efficient than Model Checking, especially for systems with many multicore processors operating concurrently. The experiments with the model implementation showed the applicability of the proposed approach in practice to real scale systems.

In future work, we plan to extend our components models library with more models of core and task schedulers and models of switched networks components. Integration with a scheduling tool which allows user-defined models of system components is also planned.

References

1. Avionics application software standard interface. ARINC specification 653. Aeronautical Radio. Annapolis (1997).
2. AUTOSAR. Enabling Innovation, http://www.autosar.org/
3. Obermaisser, R. et al.: DECOS: an integrated time-triggered architecture. Elektrotech. Inftech. 123(3), 83–95 (2006). doi:10.1007/s00502-006-0323
4. Marinescu, S. et al.: Timing analysis of mixed-criticality hard real-time applications implemented on distributed partitioned architectures. In: Proceedings of 2012 17th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2012), pp. 1–4. Krakow, Poland (2012). doi:10.1109/ETFA.2012.6489720
5. Macariu, G., Cretu, V.: Timed automata model for component-based real-time systems. In: Proceedings of 2010 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems, pp. 121–130. Oxford, UK (2010). doi:10.1109/ECBS.2010.20
6. Craveiro, J. P., Silveira, R. O., Rufino, J.: hsSim: an extensible interoperable object-oriented n-level hierarchical scheduling simulator. In: Proceedings of the 3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2012), pp. 9–14. Pisa, Italy (2012).
7. Khoroshilov, A. et al.: AADL-Based Toolset for IMA System Design and Integration. SAE Int. J. Aerosp. 5(2), 294–299 (2012). doi:10.4271/2012-01-2146
8. Balashov, V.V., Balakhanov, V.A., Kostenko, V.A.: Scheduling of computational tasks in switched network-based IMA systems. In: Proceedings of International Conference on Engineering and Applied Sciences Optimization, pp.1001–1014. Athens, Greece (2014).
9. Cassez, F., Larsen, K.: The impressive power of stopwatches. In: Palamidesi C. (eds.) CONCUR 2000 — Concurrency Theory. LNCS, vol. 1877, pp. 138–152. Springer, Heidelberg (2000). doi:10.1007/3-540-44618-4_12
10. Tretyakov, A.: Automation of scheduling for periodic real-time systems (in Russian). Proceedings of the Institute for System Programming. 22, pp.375–400 (2012). doi:10.1134/S0361768813050046
11. Bengtsson, J., Y,i W.: Timed automata: Semantics, algorithms and tools. In: Desel J., Reisig W., Rozenberg G. (eds.) Lectures on Concurrency and Petri Nets. LNCS, vol. 3098, pp. 87–124. Springer, Heidelberg (2004). doi:10.1007/978-3-540-27755-2_3
12. Andre E.: Observer patterns for real-time systems. In: Proceedings of 2013 18th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), pp. 125–134, Singapore (2013). doi:10.1109/ICECCS.2013.26