Robustness Analysis for Scheduling Problems using the Inverse Method

L. Fribourg, R. Soulat
LSV
ENS Cachan & CNRS
Cachan, France
Email: {fribourg,soulat}@lsv.ens-cachan.fr

D. Lesens, P. Moro
Data Management & Software
Astrium Space Transportation
Les Mureaux, France
Email: {david.lesens,pierre.moro}@astrium.eads.net

***Abstract*—Given a Parametric Timed Automaton (PTA) A and a reference valuation for timings, the *Inverse Method* (IM) synthesizes a constraint around the reference valuation where A behaves in the same time-abstract manner. This provides us with a quantitative measure of robustness of the behavior of A around the reference valuation. We show in this paper how IM can be applied in a specific way to treat the robustness of scheduling systems. We also explain how to use the method in order to synthesize large zones of the timing parameter space where the system is guaranteed to be schedulable. We illustrate the method on several examples of the literature as well as a case study originating from an industrial design project.**

I. Introduction

The *Inverse Method* (IM) is a method for synthesizing values of parameter timings in the framework or Parametric Timed Automata (PTAs). Different from CEGAR-based methods, IM makes use of a "good" parameter valuation π_0 instead of a set of "bad" states [2]. It takes as input a PTA $\mathcal{A}(P)$, where P denotes a vector of parameters, and a reference valuation π_0. The IM method synthesizes a constraint K on the parameters (i.e., a symbolic set of parameter valuations) such that (1) $\pi_0 \in K$ and (2) for all parameter valuation π satisfying K, the trace set (i.e., the time-abstract behavior) of $\mathcal{A}[\pi]$ is the same as for $\mathcal{A}[\pi_0]$[1].

This provides the system with a criterion of *robustness* (see, e.g., [9]) around π_0.

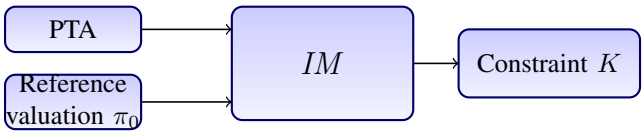

Figure 1. Functional view of IM

We explain in this paper how IM can be useful for analysing specifically the robustness of real-time systems, and how an iterative form of IM, called the Behavioral Cartography (BC) method, is useful for synthesizing schedulable zones of real-time systems.

[1]where $\mathcal{A}[\pi]$ denotes the timed automaton resulting from $\mathcal{A}(P)$ by instantiating P with π

More formally, a real-time system $\mathcal{S}$ is viewed here as a set of jobs $\{J_1, J_2, \ldots, J_n\}$. A job J_i generates a possibly infinite stream of tasks $J_{i,1}$, $J_{i,2}$, ... When a job is activated, it executes for at most time C_i, and has to terminate within the relative deadline D_i. The activation of tasks can be modeled by PTAs, where activation events are associated with transition labels. The timings (C_i, D_i) can be considered as parameters associated with each job. A *parameterized job system* $\mathcal{S}(P)$ is a set $\{J_1, J_2, \ldots, J_n\}$ associated with a vector P of parameters. Each design parameter in P can have a fixed value or be a free parameter. An *instantiated* job system $\mathcal{S}[\pi]$ is a job system $\{J_1, J_2, \ldots, J_n\}$ associated with a vector of design parameter P where each design parameter in P is assigned a fixed value according to the valuation π. For a given choice π of parameters, we say that a job J_i is *schedulable* if all the generated tasks $J_{i,k}$ finish their execution before the deadline. The system $\mathcal{S}[\pi]$ is *schedulable* if all its jobs are schedulable.

In this context, the problem of *robustness* is defined as:

Problem 1. Given a parameterized job system $\mathcal{S}(P)$ and a valuation π_0 of the parameters, find a constraint K containing π_0 such that $\mathcal{S}$ is *robust on* K, i.e.: for all $\pi \in K$, $\mathcal{S}[\pi]$ is schedulable if and only if $\mathcal{S}[\pi_0]$ is schedulable.
Remark: One is interested by synthesizing a constraint K as weak as possible.

We are also interested in the following problem:

Problem 2. Consider a parameterized job system $\mathcal{S}(P)$, and a rectangle V_0 inside the parameter space. Find the *schedulability zone* $\mathcal{Z}$, defined as the largest subset of valuations π of V_0 for which $\mathcal{S}[\pi]$ is schedulable.

We show in this paper that Problem 1 can be solved using the IM method for PTAs, and Problem 2 using BC.

The plan of this paper is as follows. We describe the related work in Section II. In section III, we explain on an example the principle of our method. In Section IV, we apply the IM-based method to various schedulability problems of the literature (jobs with variable execution

times, deadlines); we also apply the BC method in order to synthesize schedulability zones for an example of the literature; experimental results are then summarized. An industrial case study is treated in Section V. Final remarks are given in Section VI.

II. Related Work

The use of models such as PTAs and Parametric Time Petri Nets (TPNs) for solving scheduling problems has received attention in the past few years. For example, Roméo [8] performs model checking for parametric TPNs with stopwatches, and synthesizes parameter valuations satisfying TCTL formulæ. An extension of UPPAAL allows parametric model checking [3], although the model itself remains non-parametric. The approach most related to our method is [6], [7], where the authors infer parametric constraints guaranteeing the feasibility of a schedule, using PTAs with stopwatches. The main difference here relies in our choice of the Inverse Method, rather than a CEGAR-based method. First results obtained on the same case studies are uncomparable (although similar in form), which seems to indicate that the two methods are complementary. The problem of finding the schedulability region was attacked in analytic terms in [4]; the size of our examples is rather modest compared to those treated using such analytic methods. However, in many schedulability problems, no analytic solution exists (see, e.g., [12]), and exhaustive simulation is exponential in the number of jobs. In such cases, symbolic methods as ours and those of [6], [7] are useful to treat critical real-life examples of small or medium size, as examplified here in Section V.

III. Explanation of the Method

We will now explain our method on a preemptive jobshop example given in [1]. The jobshop scheduling problem is a generic resource allocation problem in which common resources ("machines") are required at various time points (and for given duration) by different tasks. We are given a fixed set M of machines. A *step* is a pair (m, d) where $m \in M$ and $d \in \mathbb{N}$, indicating the required utilization of resource m for time duration d. A *job* is a finite sequence $J = (m_1, d_1), (m_2, d_2), \cdots, (m_k, d_k)$ of steps stating that in order to accomplish job J, one needs to use a machine m_1 for d_1 time, then use machine m_2 for d_2 time etc.

The jobshop system is a set $\mathcal{S} = \{J_1, \ldots, J_n\}$ of jobs.

A. Modeling Schedulability with TAs

Consider the jobshop system $\mathcal{S} = \{J_1, J_2\}$ for 2 jobs and 3 machines m_1, m_2, m_3 with: $J_1 = (m_1, d_1), (m_2, d_2), (m_3, d_3)$ and $J_2 = (m_2, d'_2)$ with $d_1 = 3, d_2 = 2, d_3 = 4, d'_2 = 5$.

The classical problem, called the "makespan" problem, consists in finding the minimum time (makespan) needed for completing all the tasks (with the constraint that, at any time, a machine can execute only one task). In [1], it is shown how to solve the makespan problem for system $\mathcal{S}$ using a timed automaton (TA) $\mathcal{A}$.[2] Actually, we do *not* adress the makespan problem here, but rather the *schedulability* problem. More precisely, we assume given a certain bound μ, and ask whether or not the system is *schedulable*, i.e., if there is a way (schedule) to complete all the jobs within μ time units. In order to treat schedulability, we integrate into $\mathcal{A}$ a synchronized TA observer which fires a transition "DEADLINE" when the clock measuring time of the current location goes beyond μ. The system is schedulable if there is a trace (i.e. a time-abstract branch of the reachability tree) which corresponds to successive completions of all the jobs and which does not contain the DEADLINE transition.

More generally, we are going to make the following basic assumption in the rest of this paper, concerning the job system $\mathcal{S}$:

Assumption: The system $\mathcal{S}(P)$ is modeled by a PTA $\mathcal{A}(P)$ such that, for any valuation π, one can infer the schedulability of $\mathcal{S}[\pi]$ by looking at the set of traces of $\mathcal{A}[\pi]$ (i.e., by focusing on the *time-abstract* information of the reachability tree of $\mathcal{A}[\pi]$).

Under this assumption, it should be clear that the IM method provides a measure of robustness of the system around π_0. Indeed, since the set of traces of $\mathcal{A}[\pi]$ is the same as for $\mathcal{A}[\pi_0]$ for any $\pi \in K$, the system $\mathcal{S}[\pi]$ is schedulable for any $\pi \in K$ iff $\mathcal{S}[\pi_0]$ is schedulable.

B. Robustness Analysis via IM

Let us illustrate this by analyzing the robustness around the valuation $\pi_0 : \{d_2 = 2, d'_2 = 5\}$, for the bound $\mu = 10$. We first consider a parametric version of $\mathcal{A}$ where d_2 and d'_2 become parameters. We then use IM with $\mathcal{A}$ as a model input and π_0 as a valuation input. This yields the constraint K: $7 > d'_2 \wedge 3 > d_2 \wedge d'_2 + d_2 \geq 7$. See Figure 2 for a geometrical representation of K.

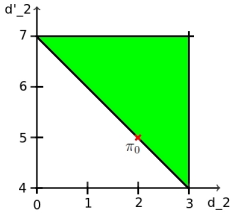

Figure 2. Geometrical representation of K around $\pi_0 : (2, 5)$.

By the IM principle, the set of traces of $\mathcal{A}$ is always the same, for any point (d_2, d'_2) of K. This set of traces is depicted under the form of a tree in Figure 9 in Appendix. We can see that there are some branches of the tree that

[2] As we consider scheduling with preemption, TAs are enriched with stopwatches and arbitrary updates [1].

do not contain any $DEADLINE$ label (these branches end at node $s73$ in Figure 9 in Appendix). These branches correspond to the schedules which are completed within $\mu = 10$ time units. The system is thus schedulable, for any point (d_2, d'_2) of K. For example, we can increase d_2 from 2 to 3, or increase d'_2 from 5 to 7 while keeping the completion time less than or equal to 10.

C. Schedulability Zone Synthesis

Let us consider a given rectangle V_0, say $[0, 11] \times [0, 11]$, and let us apply the BC method. We apply IM iteratively by letting π_0 equal to all the possible integer values of V_0. We thus synthesize different constraints K, which characterize different "behavior tiles". For any point of a tile, the behavior is uniform: either the system is schedulable (i.e., the set of feasible schedules is non empty) everywhere in the tile or nowhere in the tile. After 10 iterations, the rectangle V_0 (actually, the whole real plan) is covered by the tiles generated successively. This is depicted on Figure 3. The green (resp. red) zone corresponds to the schedulable (resp. non-schedulable) zone.

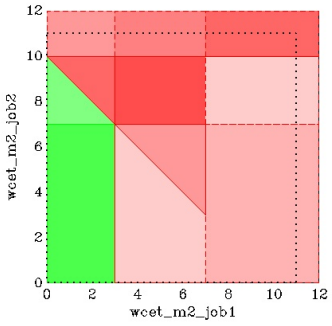

Figure 3. Schedulability zones (in green, the system is schedulable)

IV. Case Studies of the Literature

A. Jobs with Variable Execution Times [12]

The problem addressed here is the schedulability of a system of n independent job chains, denoted $J_1, J_2, \cdots, J_n$, so that they all complete within a given bound Δ when the jobs are scheduled on a processor according to a priority-driven algorithm. Roughly speaking, we let $J_{i,j}$ denote the jth job of the chain J_i. Each job $J_{i,j}$ has a fixed priority $\Phi_{i,j}$ and is preemptable. The execution time of $J_{i,j}$ is in the range $[e^-_{i,j}, e^+_{i,j}]$ with $e^-_{i,j}, e^+_{i,j}$ in $\mathbb{N}$. The release time $r_{i,j}$ of job $J_{i,j}$ is set to an integer value. $J_{i,1}$ is ready for execution at its release time $r_{i,1}$; for each $j > 1$, $J_{i,j}$ cannot execute until its immediate predecessor $J_{i,j-1}$ completes. For details, see [12].

The problem is parameterized typically by letting the $\Phi_{i,j}$s, $e^-_{i,j}$s, $e^+_{i,j}$s and Δ instantiated, and by parameterizing some of the r_is.

B. Jobs with Deadlines [6], [7]

We are given a set of jobs $\{J_1, \ldots, J_n\}$. Each job J_i is periodic of period T_i (a fixed duration of time between two activation events), and an offset O_i for its first activation time. Once a job J_i has been activated, it executes for at most time C_i and has to terminate within the deadline D_i. The system is *schedulable* if each job J_i is completed before its relative deadline D_i. [3]

We consider the case of two periodic jobs $\{J_1, J_2\}$ with $D_1 = 7, T_1 = 10, O_1 = 0, C_1 = 3, D_2 = 6, T_2 = 10, O_2 = 3, C_2 = 5$. We parameterize C_1, C_2 and O_2. Applying IM, we find the following constraint K:

$6 \geq C_2 \wedge 3 \geq C_1 \wedge 6.C_1 > 17 \wedge 2.C_1 + C_2 > 6 + O_2 \wedge$
$O_2 \geq C_1 \wedge 10 \geq O_2 + C_2$

In [6], the authors uses a CEGAR-based method to synthesize a constraint on the parameters that will guarantee that the system is schedulable. The constraint found by their method is:

$C_1 + C_2 < 6 + O_2 \wedge C_1 + C_2 < 10 \wedge C_1 + C_2 > 6 \wedge$
$C_2 < 10 - O_2 \wedge C_1 < 7 \wedge C_2 < 6$

We notice that this constraint is uncomparable with the constraint K found by IM.

C. Schedulability Zone Synthesis

Let us apply the BC method in order to determine zones of schedulability on an example with Fixed Priority ("Rate Monotonic") of [4, Section III]. There are three periodic jobs J_1, J_2 and J_3 with periods of $T_1 = 3$, $T_2 = 8$ and $T_3 = 20$ and deadlines of $D_1 = 3$, $D_2 = 8$ and $D_3 = 20$. We want to find a set of computation times C_i of each job τ_i ($1 \leq i \leq 3$) such that the system is schedulable, i.e., such that each job J_i is completed before T_i time units ($C_i \leq T_i$ for all $1 \leq i \leq 3$).

Let V_0 be the set of triples (C_1, C_2, C_3) ranging over $[0, 3] \times [0, 8] \times [0, 20]$. Algorithm BC outputs a set of tiles, and it suffices to check one point per tile to determine the schedulability of the whole tile. The result for this example is given in Figures 4, 5, 6 using a discretization step of 0.2 on V_0. The green zone corresponds exactly to the schedulability region found using analytic methods in [4] (Figure 1 (a)).

D. Summarized Experimental Results

The method IM has been implemented by tool IMITATOR. The new version IMITATOR 2.5 integrates the new features of stopwatches (in addition to standard clocks) and updates (in addition to standard clock resets), as well as powerful algorithmic improvements for state space reduction. All case studies and experiments have been performed on Ubuntu

[3] Actually, because of the periodicity of the system, we only have to be sure that it is schedulable within $lcm_{i \in \mathcal{I}}(T_i)$.

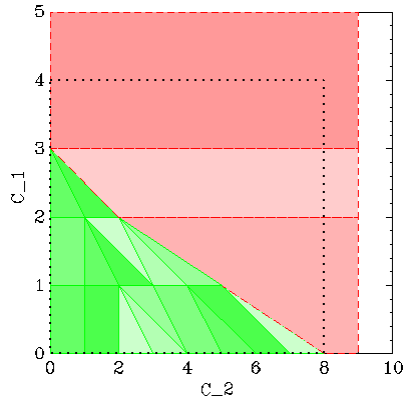

Figure 4. Schedulability zones (in green the system is schedulable)

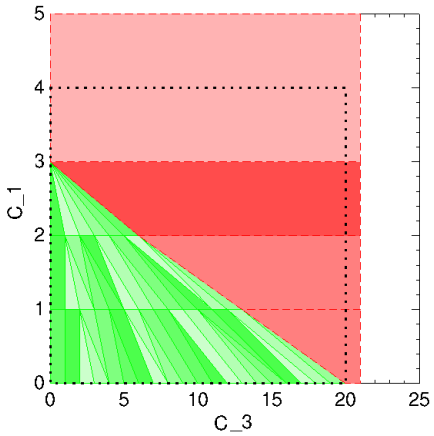

Figure 5. Schedulability zones (in green the system is schedulable)

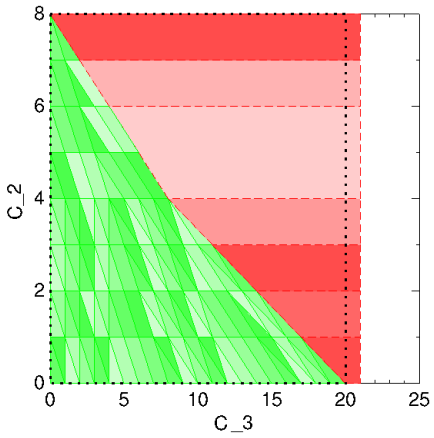

Figure 6. Schedulability zones (in green the system is schedulable)

11.10 equipped with an Intel Core 2 2.93 GiHz processor with 2 GiB RAM). In Figure 7, we give from left to right the name of the case study, the number $|X|$ of clocks of the PTA model, the number $|P|$ of parameters, the number $|S|$ of states computed, the number $|T|$ of transitions computed, the number n of iterations of the inverse method, the number $|K|$ of inequalities within the resulting constraint K, the computation time t in seconds.

Case	$\|X\|$	$\|P\|$	$\|S\|$	$\|T\|$	n	$\|K\|$	t
Examples from section III							
from [1]	3	4	53	70	10	5	0.5
LA 2×5	3	11	371	528	21	10	64
LA 3×5	3	16	4903	9043	30	5	161
Examples from section IV-A							
from [12]	15	18	215	264	15	17	85.3
Examples from section IV-B							
from [6]	6	8	676	886	15	15	289
from [7]	4	9	60	103	10	7	2.1
Examples from section IV-C							
from [4]	7	10	_	_	_	_	66

Figure 7. Experimental results

The example LA 2×5 (resp. LA 3×5) is a jobshop problem taken from the LA02 example of http://bach.istc.kobe-u.ac.jp/csp2sat/jss/, where we only consider the 2 (resp. 3) first jobs on 5 machines.

More data (including the output constraints) are given in [11] (cf http://www.lsv.ens-cachan.fr/Software/imitator/case-studies.php).

V. Industrial Case Study

A. General description

We describe here a prospective architecture for the flight control system of the next generation of spacecrafts designed at ASTRIUM Space Transportation. (This is part of a global project preparing the next generation of launcher avionics architecture [10].)

In this design, the architecture is distributed on three processors (C_{Nav}, C_{Seq}, C_{Ctrl}) devoted to the treatment of information coming from the sensors, the computational analysis of the data, and the management of data to be sent to the actuators.

The software running on each processor unit is organized into several "partitions". Each partition contains "tasks" (also often called "threads").

This is described on Figure 8: the green boxes correspond to processors, orange boxes to partitions, and blue boxes to tasks. Each task τ is periodic, and characterized by a triple (O, C, T) of timings, where O corresponds to the offset, C to the execution time, and T to the period.

Within a given partition, tasks are preemptible and scheduled according to a fixed priority scheduler, called "Rate Monotonic": the priority between two activated tasks is given to the task with the smaller period.

Tasks belonging to different partitions on a same processor are independent and can preempt each other. In case

of preemption of one task of some partition by a task of another partition, we say that there is a *partition switch*. Partition switches are performed at predefined moments of time (i.e. are time triggered). One expected output of the scheduling problem is the values of these moments (i.e., the *start time of activation* and *end time of activation* of each partition).

There are thus *a priori* several sources of nondeterminism: First, inside a same processor, there are switches of partition, second, there are the interleavings of tasks processed by the different processors.

In addition, the system is organized into *jobs* (or "end-to-end flows"): each job J_i is described as a sequence of tasks $J_{i,1}, J_{i,2}, \ldots$. ($J_{i,j}$ cannot execute until its immediate predecessor $J_{i,j-1}$ completes.) A *deadline* D_i is associated with each job J_i: the last task of the job has to be completed before the deadline.

The end-to-end flow is depicted on Figure 8 using the sequence of red arrows.

Note that, by comparison, the architecture of the flight control system running presently on the ASTRIUM Space Transportation spacecrafts is generally monoprocessor and mono-partition (see, e.g., [5]).

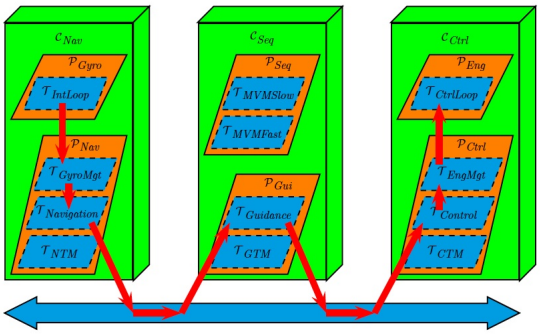

Figure 8. Architecture Scheme

B. Valuation of the parameters

A classical valuation π_0 of the triple (O, C, T)[4] can for instance be:

$(0, 8, 20)$ for task $Intloop$ (Integration Loop)
$(8, 20, 200)$ for task $GyroMgt$ (Gyroscope Management)
$(16, 8, 40)$ for task $Navigation$
$(2, 10, 50)$ for task NTM (Navigation Telemetry)
$(2, 60, 200)$ for task $MVMSlow$ (Mission & Vehicle Management)
$(0, 4, 20)$ for task $MVMFast$ (Mission & Vehicle Management)
$(117, 40, 20000)$ for task $Guidance$
$(1, 60, 200)$ for task GTM (Guidance Telemetry)
$(0, 5, 20)$ for task $CtrLoop$ (Control Loop)
$(15, 12, 50)$ for task $EngMgt$ (Engine Management)
$(1, 15, 100)$ for task $Control$
$(50, 25, 200)$ for task CTM (Control Telemetry)

The job (end-to-end flow) J considered corresponds to the list ($IntLoop$, $GyroMgt$, $Navigation$, $Guidance$, CTM, $Control$, $EngMgt$, $CtrLoop$).

The associated deadline is $D = 300$.

[4] O denotes the offset, C the execution time, and T the period.

C. Quantitative robustness analysis

We analyse the system from a quantitative robustness point of view following two steps.

In a first step, we use IMITATOR on the system instantiated with valuation π_0 using standard reachability analysis and generate all the feasible schedules that satisfy the deadline D. Among them, we focus on a schedule that minimizes the number of partition switches. This schedule is depicted on Figure 10 in Appendix, under the form of a chronogram. One division of the time corresponds to 1 time unit, and the job is completed after 285 time units. The upper level of orange rectangles indicates the running of the MVMFast-MVMSlow partition while the lower level indicates the running of the GTM-Guidance partition on processor $\mathcal{C}_{Seq}$. As already mentioned, the switches are performed at predefined moments. For example, the admissible schedule depicted on Figure 10 in Appendix can be seen as a manner for programming the partition switches on the second processor $\mathcal{C}_{Seq}$ (corresponding to frontiers between contiguous orange rectangles) at times: $51, 60, 64, 96, 104, 136, 144, 171, 251, 260, 264$.

In a second step, we apply our IM-based method to analyse the robustness of the execution times and offset times while keeping the above time-triggered sequence of partition switchings. This is done by imposing the values of time of partition switches, as specified above, and parameterizing all the execution times Cs and offsets Os corresponding to tasks $MVMFast$, $MVMSlow$, GTM and $Guidance$. The IM method then outputs the following constraint K:

$$
\begin{array}{l}
4 \geq C_{MVMFast} > 1 \\
\wedge\ 8 \cdot C_{MVMFast} + C_{MVMSlow} > 71 \\
\wedge\ C_{GTM} > 57 \\
\wedge\ 100 \geq C_{Guidance} + C_{GTM} > 89 \\
\wedge\ 120 > O_{Guidance} \geq 55 + C_{GTM} \\
\wedge\ O_{Guidance} > C_{MVMFast} + C_{MVMSlow} \\
\wedge\ C_{MVMFast} > O_{MVMSlow} > O_{GTM} > 0 \\
\wedge\ O_{MVMFast} = 0
\end{array}
$$

For any tuple of values satisfying the constraint K, the time-triggered schedule of Figure 10 in Appendix is still valid.

In more classical approaches, tools only compute a solution of the scheduling problem. The engineers using this automatically computed result may then completely loss the

feeling of their system; in particular, they cannot quantify the robustness of the design with respect to, for instance, small variations of worst case execution times or delay deadlines. In contrast, the constraint K indicates clearly to the designer some degrees of freedom, allowing a better mastering of the margin policy.

VI. Final Remarks

As shown on different case studies of the literature, our IM-based procedure provides us with a uniform method for evaluating quantitatively the robustness of scheduling solutions.

Furthermore, as examplified on an industrial case study, our approach seems able to manage a large scope of industrial problems in the domain of critical embedded software. Compared to classical approaches, it automates a boring error-prone manual activity and it formalizes the margins of evolutions of the system (margins which are generally only estimated without formally insurance of validity).

However, in spite of first promising successes, our approach meets a combinatory explosion problem when faced with even more sophisticated space systems designs that integretate more partitioning and distributed computing. We are currently working on adaptations of the method in order to tackle such highly distributed-computing architectures.

Acknowledgment

This work has been partially supported by Institut Farman (project ROSCOV).

References

[1] Y. Abdeddaïm and O. Maler. Preemptive job-shop scheduling using stopwatch automata. In *TACAS*, pages 113–126, 2002.

[2] É. André, T. Chatain, E. Encrenaz, and L. Fribourg. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science*, 20(5):819–836, 2009.

[3] G. Behrmann, K.G. Larsen, and J.I.Rasmussen. Beyond liveness: Efficient parameter synthesis for time bounded liveness. In *FORMATS*, pages 81–94, 2005.

[4] E. Bini and G.C. Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Trans. Computers*, 53(11):1462–1473, 2004.

[5] O. Boudillet, D. Dalemagne, and T. Peron. Is integrated modular avionic a solution for ATV like spacecraft control. In *Proc. 4th IAASS Conf.*, Huntsville, Alabama, USA, Sepember 2010.

[6] A. Cimatti, L. Palopoli, and Y. Ramadian. Symbolic computation of schedulability regions using parametric timed automata. In *RTSS*, pages 80–89, Washington, DC, USA, 2008. IEEE Computer Society.

[7] T.T.H. Le, L. Palopoli, R. Passerone, Y. Ramadian, and A. Cimatti. Parametric analysis of distributed firm real-time systems: A case study. In *ETFA*, pages 1–8, 2010.

[8] D. Lime, O. H. Roux, C. Seidner, and L.-M. Traonouez. Romeo: A parametric model-checker for Petri nets with stopwatches. In *TACAS*, volume 5505 of *LNCS*, pages 54–57. Springer, 2009.

[9] Nicolas Markey. Robustness in real-time systems. In *SIES*, pages 28–34. IEEE, 2011.

[10] D. Monchaux, P. Gast, and J. Sangare. Avionic-X: A demonstrator for the Next Generation Launcher Avionics. In *Embedded Real-Time Software and Systems (ERTS 2012)*, Toulouse, France, February 2012.

[11] Romain Soulat. Scheduling with IMITATOR: Some case studies. Research Report LSV-12-05, Laboratoire Spécification et Vérification, France, March 2012. Available on www.lsv.ens-cachan.fr/Publis/RAPPORTS_LSV/PDF/rr-lsv-2012-05.pdf.

[12] J. Sun, M.K. Gardner, and J.W.S. Liu. Bounding completion times of jobs with arbitrary release times, variable execution times, and resource sharing. *IEEE Trans. Softw. Eng.*, 23:603–615, 1997.

APPENDIX

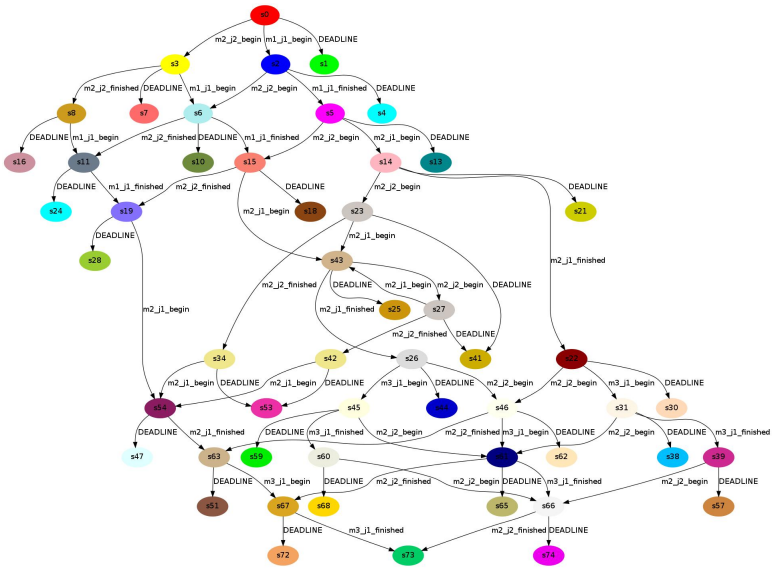

Figure 9. Trace set for the jobshop example

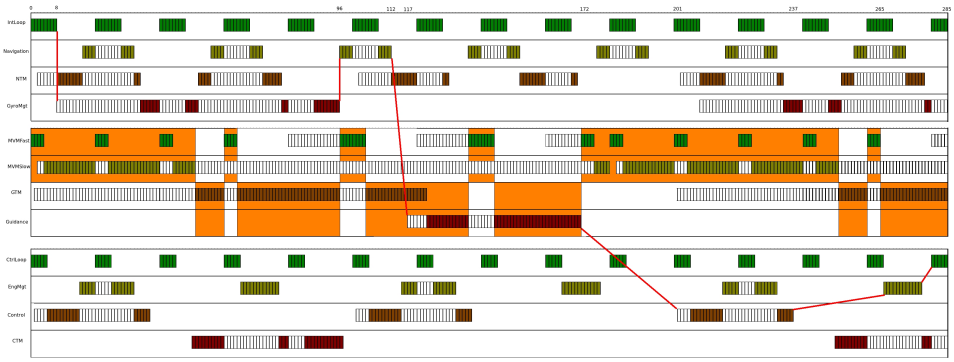

Figure 10. Chronogram of a schedule for J