ISSN 2047-3338

Scheduling with RATA Model

Farid Arfi, Jean-Michel Ilié and Djamel-Eddine Saidouni

***Abstract*— In this paper, a timed model called Resource Allocation Timed Automata (RATA) is considered, this model is constructed in compositional manner. Compared to the operational construction, this approach improves the scheduling in both time and memory space. We present two algorithms for finding the shortest paths applied to the RATA model in the case of job-shop problem. A set of benchmarks help us to experiment the efficiency of our approach.**

***Index Terms*— Scheduling, Maximality Semantics, Timed Model, Job-Shop**

I. Introduction

SCHEDULING problem is a paradigm of optimization and constraint satisfaction problems which has interested researchers over the last decades. In this area, there are well-studied methods where extensions of Timed Automata (TA) are widely applied in many contexts, based on the construction of scheduled system guided by some desired properties. One can cite; the schedulability analysis of real time distributed systems [1], timed automata with a task [2] and the job-shop problem [3], [4], [5].

In this paper, we propose to capture job-shop scheduling problems from a very intuitive and compact *description model*, called Resource Allocation Timed Automata (RATA). This model inherits from the DATA model which introduces true concurrency semantics to deal with concurrent events [6]. Extensions are provided to explicitly represent the resource requirements needed for scheduling analysis [7]. In this model, the parallelism is implicitly expressed from the starting events of actions (i.e. once started, the actions are assumed to behave in parallel until their terminations). This avoids splitting the description of running actions in start and end events, as this is proposed in the TA dedicated to scheduling problems, e.g., [8]. As another interest, the RATA reachability graphs are generally much smaller than the TA ones. However, both suffer from the well-known combinatorial explosion problem.

A standard way to attack this explosion problem consists in restraining the execution of actions by focusing on the immediate runs, to study the scheduling problems.

However in practice, other *reduction techniques* must be exploited. The main contribution of this paper consists of constructing the RATA model in a compositional manner, avoiding the costly construction method based on the operational semantics. The second contribution, consist of using set of search space reduction techniques which can be applied on the model of RATA.

The rest of the paper is organized as follows. In section II we recall the job-shop scheduling problem. The RATA model and the modeling of the job-shop problem with this model using a compositional approach of construction are given respectively in sections III and IV. Section V introduces the metrics applied on the model able to perform the scheduling. Algorithms with reduction techniques over the RATA model are proposed in Section VI. In Section VII, experimental studies are presented to highlight the efficiency of our approach. We conclude on the benefit of the RATA approach and outline some perspectives in Section VIII.

II. Job-shop problem

Job-shop scheduling is an optimization problem in which ideal jobs are assigned to resources (machines) at particular times. It is NP-complete, meaning that there is probably no efficient procedure for exactly finding shortest schedules for arbitrary instances of the problem.

The job-shop problem is described as follows. Given a finite set $J=\{j_1,j_2,\ldots,j_n\}$ of jobs to be processed on a finite set $M=\{m_1,m_2,\ldots,m_k\}$ of machines. Each job j_i is a finite sequence of actions[1] to be run one after the other, such that each action is processed without preemption, on a dedicated machine m and for a fixed time duration d, with $m \in M$ and $d \in N$. Each machine can process only one action at a time and due to precedence constraints, at most one action of each job may be processed at any time.

Definition 1: A problem instance $P=(A,\prec,M,\mu,d)$ in job-shop scheduling consists of a set $A=\{a_1,a_2,\ldots,a_m\}$ of actions, a strict partial-order precedence relation $\prec$ on A, a set $M=\{m_1,m_2,\ldots,m_k\}$ of machines, a function $\mu: A \rightarrow M$ assigning machines to actions and a duration function $d: A \rightarrow N$.

Definition 2: A schedule for a problem $P=(A,\prec,M,\mu,d)$ is determined by the function $st: A \rightarrow R^+$ indicating the start time

F. Arfi is with MISC Laboratory, Computer Science Dept., University of Mentouri, 25000 Constantine, Algeria; (Email: arfi_f@hotmail.com).

J. M. Ilié is with LIP6 laboratory and with computer Science Dept., university Paris Descartes, France; (Email: jean-michel.ilie@lip6.fr).

D. E. Saidouni is with MISC Laboratory, Computer Science Dept., University of Mentouri, 25000 Constantine, Algeria; (e-mail: saidouni@misc-umc.org).

[1] *In the context of job-shop problem, "action" means "task".*

of each action. In a deterministic setting, the end time of an action is given by $en(a)=st(a)+d(a)$. A schedule is feasible if it satisfies:

- Precedence: For every $a,a' \in A$, $a \prec a' \Rightarrow en(a) \leq st(a')$.
- Mutual exclusion: For every $a,a' \in A$, $\mu(a)= \mu(a') \Rightarrow [st(a),en(a)] \cap [st(a'),en(a')]=\emptyset$.

The length of a given schedule is defined by $\text{Max}\{en(a):a \in A\}$. An optimal schedule is a schedule whose length is minimal.

For sake of intuitiveness, we propose to specify job-shop problems, by using an algebraic language like Basic LOTOS [9]. The syntax of this language is defined as follows, where "|||" is the *parallel* operator between jobs (*J*) and ";" is the *prefix* operator expressing the sequentiality of actions, hence the precedence relation. In the proposed language, the allocation of some specific machine *m* for the execution of any action *a* is specified as "a{m}" :

$E ::= J \mid E|||E$

$J ::= stop \mid a\{m\};J$

So, a system is seen as a set of parallel jobs, each one composed of a sequence of actions. To comply with job-shop specifications, Basic LOTOS is extended in such a way that actions are mapped to specific machines. Then, if job j_i is defined by the sequence $a_1 \prec a_2 \prec \ldots \prec a_p$, its behavior expression is $a_1\{\mu(a_1)\};a_2\{\mu(a_2)\};\ldots;a_k\{\mu(a_p)\};stop$. In the following, the duration of action *a* on the machine *m* ($m=\mu(a)$) is denoted $\tau(a,m)$, instead of d(a).

III. RATA MODEL

The RATA model re-uses DATA concepts, in particular the non-atomicity of actions is captured by the fact that each transition only corresponds to a start of an action. From state to state, one or several independent actions can be launched, therefore, each state can be associated with a set of launched actions. In the model, each of these actions are represented by means of a distinct clock, dynamically created and initialized to 0 at the transition which starts the action. A set of temporal constraints is also associated with each state, expressing the conditions of ends concerning the launched actions in the state. As an example, consider the DATA of Figure 1.*a*, modeling the behavior of the process *S* already presented in the introduction. Since the actions *a* and *b* can run concurrently, a distinct clock is assigned to each one, *x* and *y* respectively. Starting from the initial state s_0, there are two possible transitions: $s_0 \xrightarrow{a,x} s_1$ and $s_0 \xrightarrow{b,y} s_2$. A label (*a*,*x*) attached to a transition indicates that the action has just been launched, and that the clock *x* will give the time spent since the launching of *a*. Similarly in the reached states, the following two transitions $s_1 \xrightarrow{b,y} s_3$ and $s_2 \xrightarrow{a,x} s_3$ are possible.

In the initial state s_0, the set of temporal constraints is empty because none of the actions is running in this state. In s_1, $\{x \geq 10\}$ specifies that the action *a* finishes its execution as soon as *x* reaches *10*. Similarly, s_2 is labeled by $\{y \geq 12\}$. In the state s_3, actions *a* and *b* can continue their runs in parallel, and each one can finish only if its proper clock reaches a value equal to its duration, so the associated set of temporal constraints is $\{x \geq 10, y \geq 12\}$.

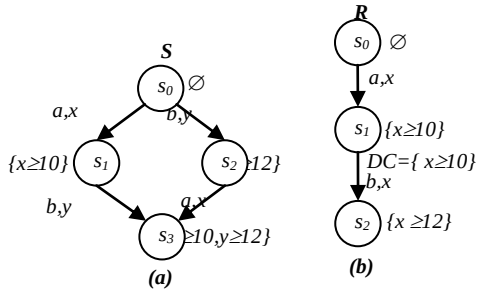

Figure 1. Behaviors of two systems in terms of DATA

The precedence relation between actions implies to annotate each transition with some additional guard, namely *Duration Condition* (*DC*). This guard on a transition expresses that the new launched action is possible provided the precedent launched ones have been terminated. In the DATA model, this is formally expressed as a subset of the set of temporal constraints attached to the source state of the transition. Consider for instance the system *R* wherein the action *a* is followed by the action *b*. The behavior of *R* is shown in Figure 1.*b*. Since at most one action can run at a certain point, the same clock *x* can be assigned to both actions *a* and *b*. from the initial state, the unique transition expresses that *a* can be launched without any duration constraint, hence $DC=\emptyset$ for this transition (not represented in the figure since empty). From the state s_1 where the temporal constraints is $\{x \geq 10\}$ for a, the action *b* can obviously be run only if the action *a* finishes its execution. This condition is expressed by the set $DC=\{x \geq 10\}$ attached to the transition which launches the action *b*. So, *b* can start at any time in the enabling open interval $x \in [10,+\infty[$.

A. Intuition of the RATA model

The RATA model is an extension of the DATA one, assuming an execution platform of (*M*) machines. An action is executed on a predetermined machine, inducing a duration for its execution. Since a machine cannot be allocated to several actions at the same time (state), a mutual exclusion mechanism must hold constraining the execution of actions.

Let us consider the system *S* again, but assume that the execution platform is either P_1 or P_2. The first one contains two machines m_1 and m_2, used for executing the actions *a* and *b* respectively, whereas the second contains a single machine *m* used for executing any action. The corresponding behaviors for *S* are represented by the RATA of Figures 2.*a* and 2.*b*, respectively.

It appears that DATA and RATA have the same structure, however the duration of the launched actions are now expressed by using the function τ such that $\tau(a,m)$ yields the duration of any action *a* executed on a machine *m*. In addition to *DC*, each transition will be labeled by another guard, namely *Availability Condition* (*AC*), expressing the mutual exclusion constraints on shared machines. As for *DC*, The condition *AC* for a transition is a subset of the temporal constraints of the source state, however concern the ones related to the machine of the transition. As usual, *AC* is not displayed when empty.

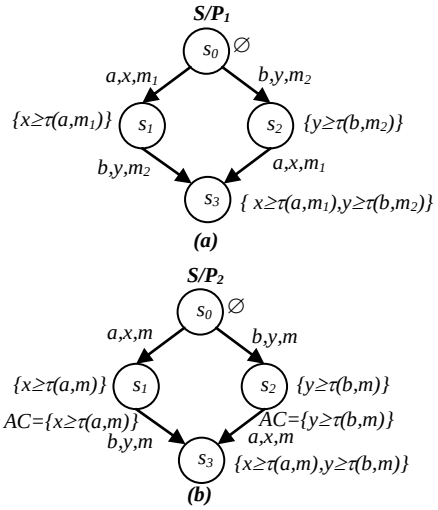

Figure 2. System *S* executed on platforms P_1 and P_2

In Figure 2.*a*, all the transitions have an empty *AC* since there is no shared machine on P_1. In Figure 2.*b*, the transition starting from the state s_1 is labeled by $AC=\{x\geq\tau(a,m)\}$. Indeed, the temporal constraint $x\geq\tau(a,m)$ in the source state relates to the shared machine *m* of the platform P_2. So in the state s_1, the action *a* is possibly in execution on *m* and the launching of the action *b* is enabled only if the temporal constraint $x\geq\tau(a,m)$ holds (i.e. the machine *m* has finished the execution of *a* and can start *b*). Observe that similar reason implies the label of the transition starting from s_2.

It is worth noting that in the RATA model, the set of temporal constraints attached to a state does not necessarily represent the parallelism of actions in the state. For instance, the state s_3 of Figure 2.*b* represents different situations of execution where at most one action can be running in s_3, with regard to the set of temporal constraints associated with s_3.

Further, *DC* and *AC* set are removed from the figures since they can be easily deduced from the clock and machine used in the label of transitions, together with the information of the temporal constraints of the source and target states.

B. Formalization

Definition 3: Let $H=\{x,y...\}$ be a set of clocks whose values are defined in a time domain R^+ and *M* a set of machines. The set $\Phi(H)$ of temporal constraints γ over *H* is defined by the syntax $\gamma::=x\geq t$, and *t* given by $\tau:A\times M\rightarrow N$ is the duration function, such that $\tau(a,m)$ represents the duration of action *a* of *A*, running on a machine *m* of *M*. Given *F* a set of constraints, its subset F_x and F^m respectively represent the constraint to the clock *x* and the different constraints related to the machine *m*.

A valuation *v* (of the clocks) of *H* is a mapping which assigns each clock of *H* to a value in R^+. The set of all valuations for *H* is denoted $\Xi(H)$. A valuation $v\in\Xi(H)$ satisfies a temporal constraint $\gamma=(x\geq t)$ with $x\in H$, which is denoted $v|=x\geq t$, iff $v(x)\geq t$. Further, this satisfaction is linearly extended to set of temporal constraints. W.r.t. $x\in H$, $[x\rightarrow 0]v$ denotes the valuation of *H* which assigns the value 0 to the clock *x* and accords with *v* over the clocks of $H\backslash\{x\}$.

Definition 4: A RATA model *RM* is a tuple (S, s_0, H, M, L, T) where:

- *S* is a finite set of states,
- $s_0\in S$ is the initial state,
- *H* is a finite set of clocks,
- *M* is a finite set of machines,
- $L: S\rightarrow 2^{\Phi(H)}$ is a mapping that associated with each state s, a set of temporal constraints $F=L(s)$, representing the set of actions possibly in execution in s, and
- $T\subseteq S\times A\times H\times M\times S$ is the set of transitions. A transition (s,a,x,m,s') also denoted $s\xrightarrow{a,x,m}s'$ represents a switch from the state *s* to the state *s'*, involving to start the action *a* on the machine *m* and define a clock *x* initialized to 0 to be associated with the action a.

Definition 5: W.r.t. a transition $(s,a,x,m,s')\in$T, the sets *DC* and *AC* of constraints are defined a follows:

- $DC=L(s)\backslash(L(s')\backslash L(s')_x)$
- $AC=L(s)^m$

The first equation comes from $L(s')=(L(s)\backslash DC)\cup\{x\geq\tau(a,m)\}$, where *DC* can be regarded as the precedence constraints to be satisfied between the actions of a job. For the first action, *DC* is empty, otherwise it is reduced to a singleton which relates to the precedent action of a within the same job. The cardinality of *AC* can be larger than one, since there may be in *s* several actions which share the same machine.

The launching of transition (s, a, x, m, s') for a given valuation *v* is constrained by the following two conditions:

- $v\ |=DC$. The specification of a system directly corresponds to the properties of precedence over the action executions.
- $v\ |=AC$. Thus, any action executed in *s* on a machine *m* must be completed to allow the firing of the transition by the same machine.

Definition 6: The semantics of a RATA $RM=(S, s_0, H, M, L, T)$ is defined by associating with *RM*, an infinite transition system *SA* on the alphabet $A\cup R^+$. A state of *SA*, also called a configuration, is a pair $<s,v>$ where *s* is a state of *RM* and *v* a clock valuation for *H*. A configuration $<s_0,v_0>$ is initial iff s_0 is initial in *RM* and $\forall x\in H, v_0(x)=0$. The two following rules express that two types of transitions can link the *SA* configurations, corresponding to an elapsing of time (*RA*) and an execution of an action of A (*RD*), respectively :

$$\frac{d\in\mathrm{R}^+}{\langle s,v\rangle\xrightarrow{d}\langle s,v+d\rangle}(RA)\qquad\frac{(s,a,x,m,s')\in T\quad v\ |=AC\cup DC}{\langle s,v\rangle\xrightarrow{a}\langle s',[x\mapsto 0]v\rangle}(RD)$$

According to the model semantics, the label *a* in the *RD* rule implies the start of an action *a* and not the whole execution of *a*. This rule can be applied only in case both sets *DC* and *AC* are satisfied. Otherwise, the time step rule *RA* is applied.

By applying the above rules from the initial configurations, we are able to compute the set of reachable configurations. Further, a run is a path of reachable configurations, by application of the two former rules. A possible run denoted $(s_0,v_0) \xrightarrow{d} (s_0,v_1) \xrightarrow{a} (s_1,v_2)$, where d represents the time spent in (s_0,v_0) and a the action to be started from (s_0,v_1), induces that $v_1=v_0+d$ and there are a machine m and a clock x such that (s_0,a,x,m,s_1) is a transition of the RATA, moreover, $v_2=[x\rightarrow 0]v_1$.

IV. MODELING WITH RATA MODEL

Unlike the approach presented in [7], [10] where the model is based on a purely operational manner of generation from its specification, we propose in this work a process of generation given in two steps. The first step is given from the operational semantics on the behavior expression of each job, which leads to generate for each job its RATA model. Then, to obtain the RATA model representing the whole system we need to compose the RATA model for the individual jobs. The major benefit of this approach is both in time and memory during the generation process compared to the first approach. This is due to the fact that once the sub-models are generated; the remaining process is related to symbolic states codifying behavior expressions.

Definition 7 (RATA model for a job): For every job $j=a_1\{m_1^j\};a_2\{m_2^j\};\ldots;a_n\{m_n^j\};stop \in J$ its associated a linear RATA model $RM^j=(S^j,s_0^j,\{h^j\},M,L^j,T^j)$ *such that:*

- $S=\{s_0^j, s_1^j,\ldots, s_n^j\}$
- s_0^j is the initial state
- $\{h^j\}$ is a singleton of clocks
- M is the finite set of shared machines where $m_i^j \in M$ for $(i=1,\ldots,n)$,
- $L^j(s_i^j)=\varnothing$ if $i=0$, $L^j(s_i^j)=\{h^j \geq \tau(a_i, mj_i)\}$ otherwise, and
- $T^j=\{(s_i^j,a_{i+1}, h^j, mj_{i+1}, s_{i+1}^j)\}$ for $(i=0,\ldots,n-1)$.

Note that one action may be in execution at any time due to the sequentiality, so at most one temporal constraint is given at any state, and one clock h^j is sufficient to measure the time. As a generating example, consider a job-shop system R running over the set of machines $M=\{m_1,m_2\}$ and consider the two jobs: $j_1=a \prec b$ and $j_2=c$, such that $\mu(a)=\mu(c)=m_1$, $\mu(b)=m_2$, $\tau(a,m_1)=4$, $\tau(b,m_2)=5$ *and* $\tau(c,m_1)=3$. The system is specified in our language by a parallel composition of two jobs j_1 and j_2: $a\{m_1\};b\{m_2\};stop|||c\{m_1\};stop$. The behavior of the two jobs is concisely represented by the RATAs of Fig. 3.

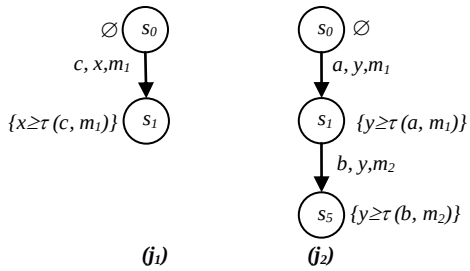

Figure 3. RATAs of jobs j_1 and j_2.

Definition 8 (parallel composition): Let $RM^j=(S^j, s_0^j, \{h^j\}, M, L^j, T^j)$ be the RATA model corresponding to each job j_i $(i=1\ldots n)$. Their parallel composition is the automaton $RM=(S, s_0, H, M, L, T)$ such that:

- $S=S^1 \times S^2 \times \ldots \times S^n$,
- $s_0=(s_0^1, s_0^2, \ldots, s_0^n)$,
- $H=\{h^1\}\cup\{h^2\}\cup\ldots\cup\{h^n\}$,
- M is the finite set of shared machines,
- $L(s)=L^1(s^1) \cup L^2(s^2) \cup\ldots\cup L^n(s^n)$ the staying condition for a global state $s=(s^1,s^2,\ldots,s^n)$ and,
- T the set of transitions contains all the tuples of the form $((s^1,\ldots, s^j,\ldots,s^n),a,x,m,(s^1,\ldots,r^j,\ldots,s^n))$ such that $(sj,a,x,m,rj)\in T^j$ for some job j.

Note that the firing conditions *AC* and *DC* which preserve the mutual exclusion of machines and the precedence relation of actions are expressed as usual from the source state, target state and the clock of the corresponding transition but over the generated model.

Take again the previous example of two jobs j_1 and j_2, the global automaton obtained by composing j_1 and j_2 is depicted in Fig. 4 (for conciseness, notice the renaming of states in the resulting model).

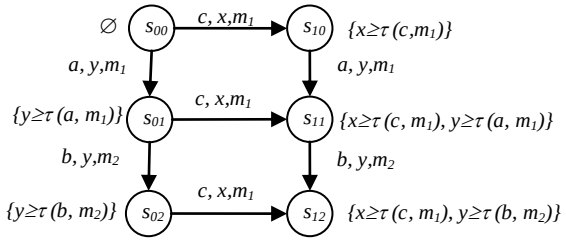

Figure 4. *The global RATA model for two jobs*

V. SCHEDULING USING RATA MODEL

In the RATA model, a run is complete if it starts from the initial state and leads to a final one, wherein the actions potentially running are considered as terminated. From every *complete run* C_R, a schedule can be straightforwardly derived, where $st(a_i)$ is the time corresponding to the start of the transition a_i in C_R. The length of the schedule coincides with the metric length of C_R. In order to compute the length of a run and the start times of the actions, one can augment the considered RATA with some additional clock used to measure the elapsed time from the beginning of the run, therefore this clock is never reset to zero. Further, its valuation is denoted t_A. Clearly, a configuration (s,v) is reachable within the time t_A iff (s,v,t_A) is reachable in the augmented RATA.

Because the time spent on states is left unrestricted by the rule *RA* in Definition 6, it appears that each qualitative path in a RATA features an infinite number of runs. This can be enhanced through the restricted notion of immediate runs.

Definition 9: (*Immediate Run*) an immediate run is a run in which whenever a transition is taken in a configuration, it is

taken as soon as it is enabled. A non-immediate run is defined as a run with the fragment: $(s,v) \xrightarrow{t} (s,v+t) \xrightarrow{a} (s',v')$, where transition taken at $(s,v+t)$ is already enabled at $(s,v+t')$ with $t'<t$.

Clearly a schedule derived from a non-immediate run exhibits laziness, therefore in order to find an optimal schedule for a given path, it is sufficient to explore the corresponding unique immediate run w.r.t. this path. The restriction to immediate runs transforms the RATA semantics into a *discrete directed acyclic graph* of configurations.

Corollary (Job-Shop Scheduling and RATA model): The optimal job-shop scheduling problem can be reduced to the problem of finding the shortest immediate run from the path of a RATA.

Let us consider a job-shop system of Figure 4. Starting from the initial configuration, the immediate runs are directly obtained from the paths of the RATA, by evaluating the satisfaction of the sets *DC* and *AC* in each reached configuration, in order to start the next actions as soon as possible (immediate execution). These evaluations require replacing each occurrence of the function τ by its corresponding value, in order to compare with the values taken by the clocks.

Figure 5 shows the derivation tree obtained by the immediate runs for the system of Figure 4, the optimal schedule is of length 9 and corresponds to the two left immediate runs of the Figure 5. In this figure, Each configuration is of the form $(s,v(x),v(y),t_A)$, where *s* represents a reachable state; $v(x)$ and $v(y)$ are respectively the valuation of the used clocks *x* and *y* in the configuration; t_A is the value of the additional clock, hence corresponds at each reachable configuration w.r.t. some run, to the time spent to reach the configuration. For sake of concision since immediate runs are concerned, there is no explicit representation of time transition. In fact, from a given configuration, the elapsing of time to start an action is implicitly considered by the evolution of clocks in the target configuration. This elapsing time is calculated according to the minimum time necessary to satisfy the set of enabled condition (for clarity this set is explicitly represented in the graph). Finally, observe that the last additional transition to an additional state *f* in every run corresponds to a hidden action, clock and machine ($\varepsilon \notin A$, $\alpha \notin H$ and $\zeta \notin M$), with an enabled condition used to terminate the execution of all of the actions not yet finished. So, the value of t_A in the final configuration of a run represents the total duration of the run, hence the length (time) of the schedule.

VI. Shortest Immediate Paths in RATA

We now describe how to improve the forward reachability algorithm to compute the shortest path in a job-shop RATA model. Since we are only interested in optimal runs, we can apply reductions mechanism in the algorithm that do not preserve all runs, but still preserve the optimal ones. We use a variant of the standard forward reachability algorithm specialized for acyclic graphs with two possible improvements.

A. Domination test

This test is used to avoid exploring identical configurations or configurations that are obviously worse than already computed ones. The domination test is based on the following definition:

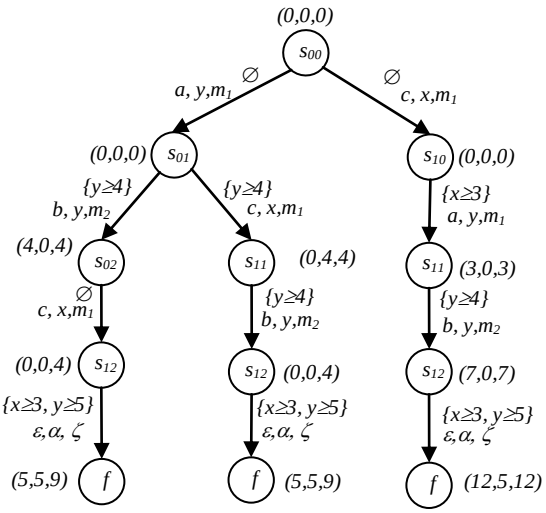

Figure 5. *The immediate runs of the RATA of Figure 5*

Definition 10: Let (s,v,t_A) and (s,v',t_A') be two reachable configurations. We say that (s,v,t_A) dominates (s,v',t_A') if $t_A \le t_A' \wedge v \ge v'$.

Clearly if (s,v,t_A) dominates (s,v',t_A') then for every complete run going through (s,v',t_A') there is a run through (s,v,t_A) which leads to a better solution (i.e. with a lower execution time).

We now propose a finer dominance relation based on a weaker relation between the clocks used in the compared configurations, e.g. (s,v,t_A) and (s,v',t_A'). For each clock *x*, we consider its duration denoted $\tau(a,m)$.

Definition 11 (D2) : (s,v,t_A) dominates (s,v',t_A') if :

$t_A \le t_A' \wedge$

$\forall x \in H, (v(x) + (t_A' - t_A) \ge v'(x)) \vee (v(x) + (t_A' - t_A) \ge \tau(a,m))$.

So, we admit that *v* could be less than *v'* for some clocks *x* in two cases.

- Either the clock difference $v'(x)-v(x)$ is compensated by the value $t = t_A' - t_A$ which means that if the same sequence of transitions is fired from (s,v,t_A) *and* (s',v',t_A'), reaching (s_r,v_r,t_{Ar}) and (s_r',v_r',t_{Ar}') respectively, then the above dominance rules globally still hold for the reached configurations. In case where the reached states are final, we have $t_{Ar}' \ge t_{Ar}$.
- In the 2nd term of the *or* clause, the action *a* associated with *x* is terminated within the duration $t_A'-t_A$. In this case, the valuation $v(x)$ must be excluded out of the dominance test since it cannot influence the future firing of transitions.

For sake of concision in this paper, the proof is not reported.

In Figure 5, we can now make a dominance reduction between the configurations $(s_{12},0,0,4)$ and $(s_{12},7,0,7)$. Here, each configuration is of the form $(s,v(x),v(y),t_A)$, and the actions associated with the clocks *x* and *y* are *c* and *b*, with respective durations *3* and *5*. The first configuration dominates the second since the 3 following conditions hold:

- $t_A = 4$ is less than $t_A'=7$.
- The action c terminates in time $v(x)+(t_A'-t_A) =0+3=3$, then the condition $v(x)+ (t_A'-t_A) \geq \tau(c,m)$ is satisfied.
- $v(y)+ (t_A'-t_A)= 0+3$ *is greater than* $v'(y)=0$.

Algorithm 1 (Forward Reachability)

Waiting $\leftarrow \{(s,v_0,0)\}$
Best $\leftarrow \infty$
While (Waiting $\neq \varnothing$*) do*
  Pick $(s,v,t_A) \in$ *Waiting*
  Succ $\leftarrow \{(s,v',t_A') \mid (s,v,t_A) \rightarrow (s,v',t_A')\}$
  Waiting $\leftarrow$ *dominate(Succ,Waiting)*
  If Succ $= \varnothing$ *then*
    Best $\leftarrow$ *min{ Best,Remains(v)+* t_A *}*
  Endif
  remove (s,v,t_A) *from Waiting*
Endwhile
End

The algorithm is a breadth-first search construction over the set of reachable configuration. It is based on a Waiting list used to store the configurations to be treated. The successor configurations are computed from the first element of the Waiting list. Each of these new configurations are tested by the *dominate* function with regards to the elements of the waiting list. A new configuration dominated by a waiting configuration is simply discarded. A waiting configuration which is dominated by a new configuration is replaced by the new one. A final configuration is reached if it currently treated configuration does not have any successor. In this case, we first calculate the remaining time needed to terminate all of the actions potentially running in this final configuration, by using the function Remains. Then, we add to Remains the access time t_A in order to obtain the total execution time of the run. This time is compared with the best time solution previously computed in order to update the best solution.

The Remains function for a set of actions $\{a_1,a_2,\ldots,a_k\}$ potentially running in a configuration is given by:

$\text{Remains}(v) = \text{Max}\{R_{a_m}(v)\}_{m=1}^{m=k}$, where $R_{a_m}(v)$ represents the remaining time to terminate the action a_m. This time is calculated by comparing the duration d_m and the valuation of the clock x associated with this launched action:

$$R_{a_m}(v) = \begin{cases} 0 \text{ if } [v(x) \geq d_m] \\ d_m - v(x) \text{ if } [v(x) < d_m] \end{cases}$$

B. Best first

The second improvement consists in applying a *best-first search* mechanism in order to explore the "most promising" configurations first. To this end, we need an estimation function over the configurations.

Consider any configuration (s,v) in the generated model, $R^m(s,v)$ is a lower-bound on the time remaining until the machine m completes the execution of the remaining of its actions from this configuration. Assume that in the configuration (s,v), we find through the temporal constraints of $L(s)$ that the action a_m is potentially running on the machine m.

Algorithm 2 (Best-first Forward Reachability)
Waiting $\leftarrow \{(s,v_0,0)\}$
Best $\leftarrow \infty$
$(s,v,t_A) \leftarrow$ *first in Waiting*
While (Best > $E(s,v,t_A)$*) do*
  Succ $\leftarrow \{(s,v',t_A') \mid (s,v,t_A) \rightarrow (s,v',t_A')\}$
  If Succ $= \varnothing$ *then*
    Best $\leftarrow E(s,v,t_A)$
  Else
    Waiting $\leftarrow$ *dominate(Succ,Waiting)*
  Endif
  Remove (s,v,t_A) *from Waiting*
  $(s,v,t_A) \leftarrow$ *first in waiting*
Endwhile
End

The remaining time is given by:

$R^m(s,v) = R_{a_m}(v) + \sum_{i=1}^{i=r} d_i^m$, where the last term represents the minimum remaining time necessary to achieve the actions that will be executed under the machine m.

From some configuration (s,v,t_A), the global estimation of the execution time, w.r.t. all the machines, is defined as:

$E(s,v,t_A) = t_A + \text{Max}\{R^m(s,v)\}_{m=1}^{m=k}$, where the last term yields the most optimistic estimation of the remaining time, with regards to all the machines.

The Best First Search algorithm maintains the waiting list sorted according to E and applies the domination test upon insertion in the list. It guarantees to produce the optimal path because the exploration is stopped when it is clear that the waiting configurations cannot lead to schedule better than those configurations found so far. Observe that in final configurations, t_A+Remains(v) is equivalent to $E(s,v,t_A)$.

VII. Experimental Results

We have implemented a tool in C++ to generate RATAs from our specification language, and have implemented the proposed techniques of scheduling. We test on a family of problems consisting of *a number of* independent jobs, each one with 4 actions. Table I summarizes the results obtained by comparing the size of the RATA with the approach of [3] proposed from (standard) timed automata. The column *#j* gives the number of jobs and *#ds* informs on the number of discrete states for each model. The columns *#dom* and *#bf* bring out the performance in terms of number of explored configurations, employing progressively the domination test and the best-first search mechanism.

As the number of jobs grows, we observe a drastic size reduction by using the RATA approach. In particular with regard to the domination test, the gain can rapidly reaches orders of magnitude.

TABLE I. The results for N jobs with 4 actions

#j	*Timed automata*			*RATA*		
	#ds	*#dom*	*#bf*	*#ds*	*#dom*	*#bf*
2	77	100	38	25	28	22
3	629	1143	384	125	180	105
4	4929	11383	1561	625	1251	306
5	37225	116975	2810	3125	9775	714
6	272125	1105981	32423	15625	59213	2520

We have considered also three sets of small and medium benchmarks taken from the known OR-library. Firstly three instances of size 10*5 (10 jobs and 5 machines): LA01, LA03 and LA05. Then, we considered the set of instances of size 15*5 (15 jobs and 5 machines): LA06, LA08 and LA10. As the previous instances are easily solved, we finally considered a medium instances of size 20*5 (20 jobs and 5 machines): LA11, LA13 and LA15. The latter is not resolved by our algorithm of best-first within a time limit of five minutes. The computational equipment for the experiments was a Pentium machine with 3 GHz and a Windows7 operating system.

TABLE II. THE RESULTS FOR LA PROBLEMS

instance	*#bf*	*#time*	*Opt*
LA01	176	0.1	666
LA03	3025	2.1	597
LA05	400	0.0	593
LA06	32460	11.2	926
LA08	17461	4.3	863
LA10	2851	0.4	958
LA11	13327	3.7	1222
LA13	3744	1.5	1150
LA15	/	/	/

Note that in the results given in the second table, we have used an improved version of heuristic function that calculates remaining time *E* in best-first algorithm. This heuristic is obtained by a simple modification of the Jackson's preemptive schedule [11].

VIII. CONCLUSIONS AND PERSPECTIVES

Exploiting the RATA model in order to solve optimal job-shop scheduling problems is a novel application of models based maximality-semantics, in addition to verification purpose [12], [13].

The RATA model appears to be syntactically close from the standard timed automata approach, taking into account action duration and shared resources. One of its major distinguishing features its behavioral compact representation that concentrates on the starts of the actions.

Our perspectives should be to extend the proposed algorithms in order to treat large size problems. To tackle the exponential blow up of configurations, we can reuse the ideas of [14], [15], which argue that one should minimize the length of the schedules without necessarily obtaining the optimal solution.

REFERENCES

[1] M. M. Jaghoori, F. S. de Boer, and M. Sirjani, "Schedulability of asynchronous real-time concurrent objects," Logic and Algebraic Programming, vol. 78(5), 2010, pp. 402-416.

[2] E. Fersman, P. Krcal, P. Pettersson, and W. Yi," Task automata: Schedulability, decidability and undecidability," Journal Information and Computation, vol. 205(8), 2007, pp. 1149-1172.

[3] Y. Abdeddaim, E. Asarin, and O. Maler, "Scheduling with timed automata," Theoretical Computer Science, vol. 354(2), 2006, 272–300.

[4] Y. Abdeddaim, and O. Maler, "Preemptive job-shop scheduling using stopwatch automata," in: Proc. TACAS'02, 2002, pp. 113–126.

[5] P. Sebastian, S. Olaf and, E. Sebastian, "Efficient synthesis of production schedules by optimization of timed automata," Journal of Control engineering practice, vol. 14(10), 2006, pp. 1183-1197.

[6] N. Belala and D.E. Saïdouni, "Non-Atomicity in Timed Models," in Proceedings of ACIT'2005, Al-Isra Private University, Jordan, December 2005.

[7] D.E. Saïdouni, F. Arfi, and J.M. Ilié, " Hétérogénéité dans les modèles temps-réel, " in Proceedings of Information Systems and Technologies: (ICIST'11), Tebessa, Algeria, 2011, pp. 553-561, ISBN: 978-9931-9004-0-5.

[8] R. Alur and, D. Dill, "A Theory of Timed Automata," TCS, vol. 126, 1994, pp. 183-235.

[9] T. Bolognesi, and E. Brinksma, "Introduction to the ISO Specification Language LOTOS," Computer Networks and ISDN Systems, vol. 14, 1987, pp. 25-59.

[10] F. Arfi, D.E. Saïdouni and J.M. Ilié,"A model for job-shop problem", International Conference on Information Technology and e-Services (ICITeS'12), Sousse (Tunisia), 24,25 and 26 March., ISBN: 978-9938-9511-1-0, 2012, pp. 640-645.

[11] J. Carlier and E. pinson ."An algorithm for solving the job-shop problem". Management Science, 35(2),1989,164-176.

[12] D.E. Saïdouni, A. Benamira, N. Belala, and F. Arfi, "FOCOVE: Formal Concurrency Verification Environment for Complex Systems," in Proceedings of Intelligent Systems and Automation (CISA'08), Annaba, Algeria, Vol. 1019 (1) of American Institute of Physics Conference Proceedings, 2008, pp. 375-380, ISBN:978-0-7354-0540-0.

[13] D.E. Saïdouni, and A. Ghenaï, "Intégration des Refus Temporaires dans les Graphes de Refus," in Proceedings of NOTERE'2006, Hermes, Toulouse, France, 2006.

[14] S. Yang, D. Wang, T. Chai, and G. Kendall, "An improved constraint satisfaction adaptive neural network for job-shop scheduling," Journal of scheduling, vol. 13(1), 2010, pp. 17-38.

[15] C.R. Vela, R. Varela, and M. A. González, " Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times," Journal of Heuristics, vol. 16(2), 2010, pp. 139-165.