Multi-resource sharing scheduling considering uncontrollable environment

Mahya Rahimi, Emil Dumitrescu, Eric Niel

Multi-resource sharing scheduling considering uncontrollable environment

Mahya Rahimi
AMPERE Laboratory,
INSA Lyon
Villeurbanne, France
mahya.rahimi@insa-lyon.fr

Emil Dumitrescu
AMPERE Laboratory,
INSA Lyon
Villeurbanne, France
emil.dumitrescu@insa-lyon.fr

Eric Niel
AMPERE Laboratory,
INSA Lyon
Villeurbanne, France
eric.niel@insa-lyon.fr

***Abstract*—This paper addresses the problem of scheduling through a visual, expressive and formal modeling approach, based on Timed Game Automata (TGA). The originality of the proposed approach lies in integrating various kinds of uncontrollability in the scheduling problem and also the ability of handling the sharing of multiple resources. This uncontrollable behaviors consist of the start time, the duration of the task and the failure occurrence in a schedule which are modeled by TGA. The models have the advantage of being directly exploitable by means of synthesis tools. To obtain the minimum makespan and optimal schedules as the result, a time-optimal reachability game is performed through Tiga tool. The obtained result for a scheduling example shows the huge difference in the obtained schedule with and without considering uncontrollable behaviors.**

Keywords—scheduling, timed synthesis, uncontrollability, timed game automata, makespan, multi-resource sharing

I. INTRODUCTION

Scheduling problems can benefit from significant enhancements when restricted to single-resource applications [1]. Yet in some application domains, it is essential to assign more than one resource to each task [2], [3]. Many approaches exist for solving scheduling problems; optimization methods and tools, advocated by [4], already offer competitive solutions for optimization and scheduling in particular. Yet, research issues related to modeling, such as behavioral and dynamic interaction, or expressing disruptive behaviors need further investigation. Accordingly, many research contributions advocate the use of (max,+) algebra, Petri nets or automata theory for addressing these issues. In this direction, automata models appear to be very convenient: they are intuitive, expressive, robust against changes in the parameter setting and against changes in the problem specification [5], [6], while offering a formal framework.

Automata theory and adjacent tools, such as verification approaches, have been widely used for addressing scheduling problems [5], [7], [8]. Yet few contributions take into account the multi-resource sharing (MRS) aspect. Subbiah et al. [7] consider scheduling of multi-product batch plants with sequence-dependent changeover procedures through timed automata. The authors do not take into account MRS criteria in the proposed model. G. Behrmann et al. [6] treat a type of job shop scheduling problem for lacquer production using a timed automata modeling approach. The authors use a heuristic approach to reduce the search space. They also propose solutions that are applicable for other scheduling cases. Abdeddaïm et al. [5] propose shortest path algorithms for timed automata to find optimal schedules in a job shop scheduling problem. The authors also investigate non-lazy scheduling with uncertain task duration. Panek et al. [9] integrate mathematical modeling into the reachability analysis of timed automata and propose an algorithm for scheduling and makespan minimization of job shop problems. Marangé et al. [10] propose a scheduling approach to handle reconfiguration of a manufacturing plant. Following a reconfiguration request, a scheduling is obtained for a set of items produced by a set of machines. The problem is classified as a job-shop scheduling problem which is threated by reachability analysis. All the studies mentioned above are classified into classical job-shop problems and hence do not consider MRS criteria.

It should be mentioned that there exist some related studies that in addition to using automata models, integrate MRS criteria in the problem. Whereas, they do not generate any schedule, but their goal is to only minimize the makespan [11].

Apart from MRS aspect, in all of the above studies, it is considered that task behaviors are deterministic and resources are reliable. While the real world is not like that: disruptive behaviors may exist affecting the pre-scheduled execution of tasks. Start time of tasks may be changed due to several reasons like the delay in sourcing of raw material; task processing may take more or less time than expected; resources may break down; undelayable tasks may arrive in the middle of the schedule, etc. [5], [12]–[15].

Disruptive behaviors can be integrated in automata modeling by using the notion of *uncontrollability*. Abdeddaïm et al. [5] use an extension of timed automata for solving the classical job-shop problem. The authors divide the transition set into controllable and uncontrollable subsets. They propose shortest path algorithms for timed automata to find the optimal schedules. They also investigate non-lazy scheduling with uncertainty in task duration. David et al. [16] propose a framework to model and analyze a variety of schedulability scenarios for problems that deal with multiprocessor systems, timing uncertainties in arrival and execution times, possible dependencies of tasks and preemption of resources. The problem is modeled by timed automata. Dumitrescu et al. [17] propose a framework for multi-criteria optimal controller synthesis to model and optimize fault-tolerant distributed systems considering task execution cost and its service quality. Moreover, to combine criteria, the authors consider three different methods: aggregation, hierarchization and translation. Marangé et al. [10] develop a job-shop scheduling model by communicating automata to handle reconfiguration of a manufacturing plant due to resource failure. Following a reconfiguration request, a scheduling is generated for a set of products that are produced by a set of machines. This

schedule can be obtained by means of reachability analysis on the model.

Su et al. [18] address a minimum-makespan supervisory synthesis job shop problem. They assume also occurrence of uncontrollable events such as malfunction of a component when the system reaches a certain state, being unable to execute a program immediately when the operating system is still retrieving all relevant execution resources for this program according to a pre-specified internal mechanism unknown to the end user, producing imperfect product, etc. The makespan of the problem is computed through theory of heap-of-pieces. A timed supervisory control map is also presented that is capable of implementing the synthesized minimum-makespan sublanguage. The author models the problem by weighted and un-weighted deterministic finite state automata. Boukra et al. [19] propose new representations for (max,+) automata in order to describe their extremal behaviors. Consequently, the defined automata is applied to performance evaluation application. Thereby, the worst case and optimum case of behavior of automata are formulated in a form that has polynomial complexity. Furthermore, the authors defines an equation to find the start time of actions in optimal and worst cases. Thus, if the automaton corresponds to the synchronous composition of components of a scheduling problem, the minimum makespan and start time of tasks can be found through the presented formula for the performance evaluation. In this article, the presented formulae are applied to an example of MRS scheduling problem. Supervisory control of automata and the notion of uncontrollability represents undelayable and unpreventable tasks.

According to the contributions mentioned above, the modeling of uncertain behaviors using uncontrollability appears to be of great interest. Yet, different notions of uncontrollability are explored in different specific ways.

The purpose of this work is to explore several useful notions of uncertain behaviors, model them using the uncontrollability mechanism, apply them to a specific class of problems and provide a global modeling and scheduling framework suitable for this class. The scheduling problems considered in the sequel require modeling specificities: concurrency, explicit timing, interaction mechanisms and a means of specifying uncontrollability. The *timed game automata* (TGA) model [20] provides these mechanisms, which is why it is used throughout this work. In order to do scheduling, a worst-case makespan is obtained which enables scheduling the tasks of time-critical systems.

The main contribution of this work is a novel modeling approach based on TGA in order to perform makespan minimization, by considering:

- the occurrence of different kinds of uncontrollable behaviors
- the integration of MRS concept

The remainder of this paper is as follows: Section II presents the problem description. Different types of uncontrollability are enumerated in Section 11). In Section IV, the scheduling problem is modeled through TGA. Section V discussed the solving approach for the scheduling problem. In section VI through an example, the difference between a schedule with and without presence of uncontrollable behaviors is demonstrated. Finally, section VII is devoted to the conclusion.

II. PROBLEM DESCRIPTION

The aim of this work is to find a schedule of minimum duration (minimum makespan) in order to execute a set of tasks just once. Tasks consist of either value-added operations or preventive maintenances on resources. Tasks should be performed in parallel, by respecting predefined mutual exclusion and precedence constraints. The following assumptions are expected to hold:

1) Duration of tasks may be restricted to be bounded within an interval.
2) Start time of tasks may be subject to bounded uncertainty.
3) Task preemption is not allowed, whereas due to a failure a task may be canceled.
4) There may be conflicts for performing tasks at the same time, but when there is no conflict between them, they should be performed simultaneously.
5) There may exist precedence constraints between tasks.
6) All tasks are ready to be executed at time zero.
7) Resources are pre-assigned to tasks.
8) Resources are reusable (they are not raw materials and by performing maintenances, they can be used in every cycle).
9) Each resource can be used to execute only one task at a time, but a task may use more than one resource simultaneously.
10) Resources are not reliable and may fail in two conditions: 1. during performing a task, 2. during idle time of resources
11) If resources are not broken down, they are available at time zero.

III. DIFFERENT TYPES OF UNCONTROLLABLE BEHAVIORS

In this section, four major kinds of uncontrollable behaviors are discussed.

- *Resource failure while performing a task*: In certain cases, resources are not reliable and for that reason, during execution of tasks, unpredictable failures may occur. Therefore, tasks relating to that resource cannot be executed until its reparation.
- *Resource failure when it is idle*: Not all the failures happen when performing tasks. Sometimes failures may occur when a resource is idle and therefore it won't be able to start execution of any task until it is repaired.
- *Bounded uncertainty in duration of tasks*: In real world, human intervention, incomplete information or uncertain environment may cause uncertainty in duration of tasks.

- *Bounded uncertainty in start time of tasks*: In certain applications, uncertain task performing conditions might cause a delay in start time of tasks from the moment of their release time. By definition, release time is the earliest time when a task can start execution.
  This type of uncontrollable behavior may happen to two types of resource assignment; when tasks require resource assignments from their decided start time or when resources should be assigned from the real start time of the task execution.

The inclusion of these types of uncontrollability in the model at hand is both intuitive and provides generic models.

IV. Modeling the Scheduling Problem through TGA

A schedule S over a set of tasks T can be considered as a mapping $S: \mathrm{T} \rightarrow \mathbb{R}_{\geq 0}$ assigning a starting time to each task of T. Yet, by considering uncontrollable behaviors, this representation is no longer valid, and the schedule should be replaced by a *scheduler*, featuring a dynamic behavior, interacting with an environment in order to implement a winning game: reach a final desired configuration as fast as possible, despite the behavior of the environment, considered as an adversary.

In this situation, some actions are done by the environment and therefore are uncontrollable and the others are done by the controller. For example, let's assume the case when the start time of the tasks are controllable and there is a possibility of resource failure it their idle state. To finish all the tasks in a minimum time, there is a competition between the controller and the environment. The environment prevents the controller to do the optimal actions by creating uncontrollable behaviors. This competition can be modeled by Timed Game Automata [21]. In this case, tasks can be launched by the controller through taking a controllable transition, while at the same time, the environment may take an uncontrollable transition to cause a failure. In order to perform all the tasks, the controller finds a strategy that guarantees reaching the final state whatever the opponent (the environment) is doing.

The definition of a MRS scheduling problem relies on a collection of three kinds of TGA models expressing: mutual exclusion, precedence and task launching.

Definition (MRS scheduling problem): A MRS scheduling problem statement $S = (T, E, G_{tl}, G_{pr}, d)$ consists of (1) a set $T = \{t_i | i = 1, \dots N\}$ of tasks that might be necessary to be executed simultaneously, (2) a family of mutual exclusion constraint sets $E \in 2^T$ that are modeled as a set G_{me} of Mutual Exclusion automata (ME), (3) a duration function $d: T \rightarrow \mathbb{R}_{\geq 0}$ assigning fixed durations to tasks, (4) a set G_{tl} of Task Launcher (TL) automata that model triggering of each task for a specific number of times. $|G_{tl}| = |T| = N$ which means that for executing every task, there exists one TL automaton (5) a set G_{pr} of Precedence (PR) automata modeling precedence constraints among tasks. This set could be empty.

In order to perform the scheduling, the MRS scheduling problem is first modeled by a collection of TGA. Then through a timed game analysis, the strategy for obtaining an optimal schedule and makespan is found. In the sequel, tasks are triggered using the interaction mechanisms of TGA: communication channels. Each task launcher (TL) triggers one task *t,* by emitting the communication signal "*t_s!*". The mutual exclusion (ME) and precedence (PR) models listen *"t_s?"* and react accordingly, by enabling/disabling further task triggering so that the modeled constraints are always satisfied.

It should be mentioned that in the sequel, whenever variables of a model are discussed locally, they are not indexed. For example in Fig. 1 where there are numerous tasks and tasks are not discussed locally, they are indexed as $t_i, t_j, \dots, t_z$; while in Fig. 3, one task is discussed locally and task t is not indexed.

A. The Mutual Exclusion (ME) model

A ME automaton model represents the conflict among a set of tasks $TM \subseteq T$. The model waits in its initial location l_0. When receiving a launching signal t_k_s in order to run t_k, it takes a transition from l_0 to the task location t_k and resets the clock cl. After elapsing $d(t_k)$ time units, it takes the transition from location t_k to l_0. In fact, $d(t_k)$ is the duration of task t_k. In the initial location, the automaton waits for another launching signal to execute a new task. Figure 1 indicates that tasks $TM = \{t_i, t_j, t_k, \dots, t_z\}$ are in conflict.

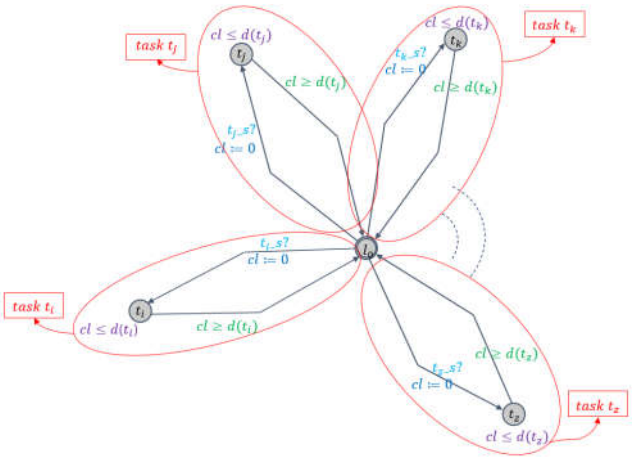

Figure 1. Modeling pattern of timed ME automaton

The ME model can be adapted in order to integrate the uncontrollable behaviors introduced in the previous section.

1) Unpredictable resource failure

In each ME automaton, due to using common set of resource, tasks can be in conflict. Therefore, if one of these resources breaks down, tasks related to that ME automaton cannot be executed until the resource is repaired. Reparation duration depends on the resource. In automaton j, this duration is denoted by $d(rep_j)$ and is equal to the maximum time needed for reparation of resources related to this automaton (Fig. 2 and Fig. 3).

Variable b_j is dedicated to modeling a bound of resource failures. It denotes the number of failures in a scheduled cycle. For avoiding schedulability problems, this number should be limited. In fact, if no limit is put for this number, there would be a possibility that a scheduled cycle would never end. Without loss of generality, it is assumed that at most one failure may occur in a cycle. Therefore when $b_j = 1$, no more failure can occur in a ME automaton.

Two kind of resource failures can be modeled in a ME automaton:

a) Failure while a resource is idle

During idle time of resources, a failure may occur. A failure is an uncontrollable action, modeled by a dashed transition. As mentioned above, for the sake of schedulability, it is assumed to be limited to occur at most once. Hence, in the model of failure (Fig. 2), if $b_j = 0$, an uncontrollable transition can be taken by the environment, the clock is reset and the automaton reaches the *fail* location. Then after elapsing $d(rep_j)$ time units, the automaton takes a transition to the initial location and updates variable b_j to one.

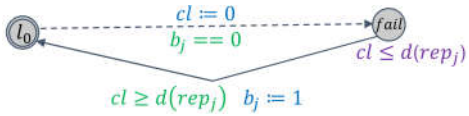

Figure 2. Modeling pattern of a resource failure in its idle time in ME automaton j

b) Failure while doing a task

In automaton j, during execution of task t, a failure may happen. The reason of the failure could be a breakdown in the resources related to the mutual exclusion set j or another mutual exclusion set that share task t.

Execution of task t happens in the location t. Therefore, uncontrollable transitions representing failures should be taken from this location. For modeling a breakdown in the resources related to the ME automaton model j, an uncontrollable transition is added from location t. If it is the first time that a failure happens in this set of resources, i.e. $b_j = 1$ and if task i is not finished yet, i.e. $cl < d(t)$, this transition can be taken by environment. By taking this transition, clock cl is reset and a signal $stop\_s$ is sent to the other model components that share task t to stop the task. Thereby the automaton reaches location stp where it waits for repairing the broken resource.

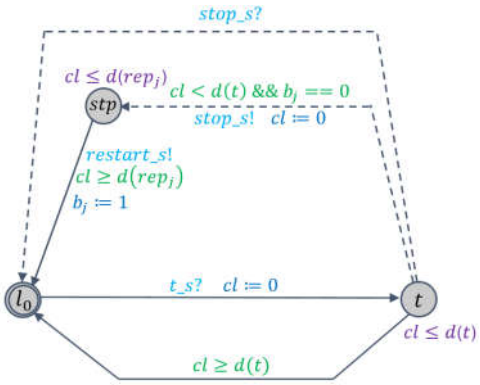

Figure 3. Modeling pattern of a task in a ME automaton subject to resource failure

After the repairing period, i.e. $d(rep_j)$, the resource is ready and all the tasks belonging to this task-conflict set, i.e. TM_j, can start execution. Therefore, the automaton takes a transition to the initial location, updates variable b_j to one to prevent more failures and sends signal $restart\_s$ to the other model components that share the same task, to enable restarting t_k. These model components can be other ME models, or precedence (PR) models automata or task launcher (TL) models, all presented in the sequel.

At the same time when a task is interrupted due to a failure in a ME automaton, it should also stop execution in other ME automata that share the same task. Hence, as depicted in Fig. 3, a transition from location t to the initial location is added to the model of a task. Whenever the automaton is in task location t and receives signal $stop\_s$ from another ME automaton, it stops execution and goes to the initial location.

Thereby, with possibility of abovementioned types of failures, the model of a ME automaton can be as presented in Fig. 4.

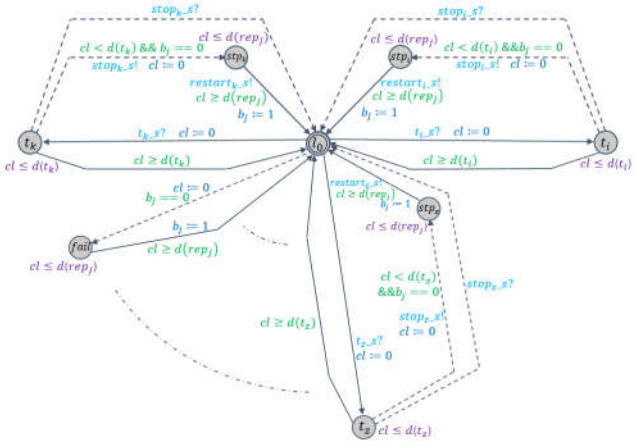

Figure 4. Modeling pattern of a ME automaton subject to resource failure

2) Bounded uncertainty in duration of tasks

In some applications, the exact duration of the tasks may not be given, but rather durations are restricted to be bounded within an interval of the form $[d_1(t), d_2(t)]$.

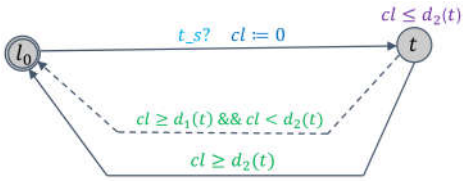

Figure 5. Modeling pattern of a task with uncertain duration in a ME automaton

Hence, when the ME automata is in the task location t, from the moment the clock reaches $d_1(t)$ till $d_2(t)$, the automata may take a transition to the initial location. Therefore, an uncontrollable transition is added to enable this uncertain movement from task location to initial location when $d_1(t) \leq cl < d_2(t)$. But as assumed, it is certain that the duration of the task is no longer than $d_2(t)$, a controllable transition is added from the task location to the initial location to be taken if the automaton still waits in task location at time $d_2(t)$ (Fig. 5).

3) Bounded uncertainty in start time of tasks

Start time of a task may be uncertain. Whereas, as mentioned previously, this uncertainty can be treated with two points of view. The first one is to keep all the necessary resources

available from the decided start time of the task until its real start time. The second point of view is to let the environment start the task during a time window while the resources are idle. The global model should represent these cases appropriately:

a) Environment starts the task in a time window whenever related resources are idle

This kind of uncontrollability does not have any impact on the model of the task in ME automaton model. For considering this type of uncertainty, only the model of TL automaton changes, as shown in the sequel.

b) Keeping resources available from the lower bound of the decision time window

Resources should be reserved from the initial moment of decision time window. Hence, from this moment, the automaton should move from the initial location to another location to prevent starting the other tasks. Although, since in this moment the task is not started yet, the ME model cannot move to the task location. For that reason, in this instant, by receiving the signal decide_s from the related TL automaton, the ME model moves to a new location $deci\ de$ When the environment decides to start the task, it sends signal t_s from the TL automaton to the ME automaton. Thereby, clock is reset and the ME automaton reaches the task location t and starts the task. After waiting $d(t)$ time units in this location and finishing the task, the automaton takes a controllable transition to the initial location and waits for starting a new task (Fig. 6).

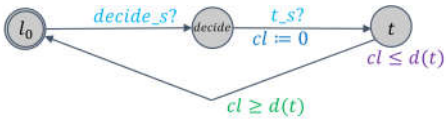

Figure 6. Modeling pattern of a task with uncertain start time in a ME automaton

B. The Task Launcher (TL) model

In order to launch every task, a TL automaton is needed and hence the number of TL automata models is equal to the number of tasks to schedule. A task launcher should be able to launch a task whenever its related task is executable in other automata. This means that the other automata containing the same task should be in locations dedicated to listening signal t_s. Thence, through sending signal $t\_s!$ by the task launcher of task t, the related ME and PR automata receive signal $t\_s?$ and trigger task t. For this purpose, in this model, function $enabled(t)$ is defined. Therefore, whenever all the ME automata sharing the task are in their initial location, the output of function $enabled(t)$ becomes true and the task can be launched. In this moment, the automaton sends a signal t_s to the other automata sharing the task t and reaches the location f (Fig. 7). Thereby, t starts execution in these automata.

Therefore the guard function $enabled: T \rightarrow Bool$, which is associated to the transition of the TL model, is true if and only if the following condition holds:

$$\forall i \in [1, M]: t \in TM_i \rightarrow \forall t' \in TM_i \backslash \{t\}: \neg pending(t', l_i) \quad (1)$$

where $pending\ TM_i \times L_{mei} \rightarrow Bool$ is defined as

$$pending(t, l_i) = \begin{cases} false, & l_i = l_0 \\ true, & o.w \end{cases} \quad (2)$$

where M is the number of ME automata, t is the name of the task, TM_i is the set of tasks engaged in ith ME automaton. In the ith ME automaton, L_{mei} is the set of locations, l_i is its current location and l_0 is the initial location. Hence, function *pending,* presented in equation (2), verifies if a ME automaton is in a location where a task t is being executed.

Hence, task t can only be enabled if all the ME components observing t are in their respective initial location l_0.

Figure 7. Modeling pattern of a task launcher automaton

The TL model should also be adapted in order to integrate the failure behaviors mentioned above

1) Unpredictable resource failure

a) Failure while a resource is idle

This kind of failure does not interrupts tasks since a TL automaton is only related to tasks and not resources. Therefore, there is no need to change the model of TL automaton for considering failure while a resource is idle.

b) Failure while doing a task

Whenever a failure occurs during execution of a task, the task should be repeated after repairing the resource. Therefore, after this period, TL should return to its initial location to be able to launch the task another time.

To add this feature to the model of TL automaton, one additional location stp is created. When a task is during execution, the automaton is in location f. Therefore, at the instant of resource failure, it receives the signal $stop\_s$ from the ME automaton in which the resource failure was triggered and reaches location stp. In this location, the automaton waits until

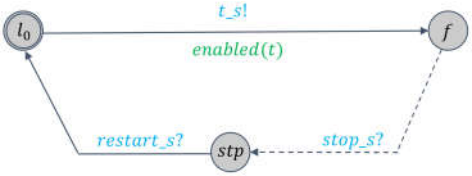

Figure 8. Modeling pattern of a launcher automaton for a task subject to failure

reparation of the related resource. After this period, the automaton receives another signal $restart\_s$ from the ME automaton to which the resource failure was related. Thereby TL automaton goes to the initial location (Fig. 8).

2) Bounded uncertainty in duration of tasks

A TL only launches a task and does not concern duration of tasks. Hence, for considering uncertain duration of tasks, there is no need to change the model of TL automaton.

3) Bounded uncertainty in start time of tasks

a) Environment starts the task in a time window whenever related resources are idle

Start time of a task may be uncontrollable such that from the instant of releasing a task, its start time vary in an interval of the form $[0, lim]$ where lim represents the maximum delay for starting the task. For modeling this feature, two issues should be

considered; the first one is that as before, in addition to the communication signal t_s, the output of $enabled$ function for the task t should be verified. The releasing time of the task is let to be time zero in TL automaton. Hence, the second issue is that from time zero to the moment prior to the time bound $li\ m$ launching the task t is uncontrollable. If the task is not launched until this time, the automaton should be forced to launch the task at this instant. Furthermore, its launching cannot be delayed more than this threshold.

Therefore, two transitions lead the automaton from the initial location l_0 to the location f. One transition is an uncontrollable transition that can be taken by the environment if $enabled(t)$ is true and the value of the local clock cl is less than $li\ m$ Another transition is a controllable transition that can taken by controller if $enable(t)$ is true and the value of cl is equal to $li\ m$ When taking one of these transitions, a communication signal t_s will be sent to the other automata that share the task t (Fig. 9).

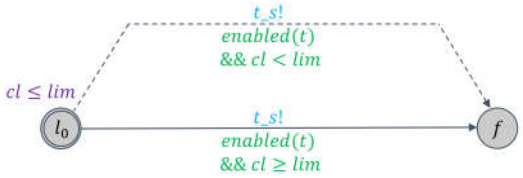

Figure 9. Modeling pattern of a TL automaton for a task with uncertain start time when the resources are occupied from the instant of execution

b) Keeping resources available from the lower bound of the decision time window

In this case, at the instant when all necessary resources for execution of task t are available, i.e. the output of $enable(t)$ is true, the TL automaton of task t reserves resources. To this end, it takes a transition to $deci\ de$ location and sends a signal $deci\ de\_s$ to the ME automata sharing task t. Therefore, the related ME automata reach to their $deci\ de$ location and this issue prevents other tasks in the related ME automata to occupy the necessary resources and to be executed. Furthermore, by taking this transition, the local clock cl is reset. In location $deci\ de$, the automaton waits for the environment to take an uncontrollable transition to the location f until the instant that the clock value reaches $li\ m$ If the environment did not took the transition, the supervisor takes a controllable transition to f at time $li\ m$ In the both cases, a communication signal t_s is sent to the other automata that share t to start the task (Fig. 10).

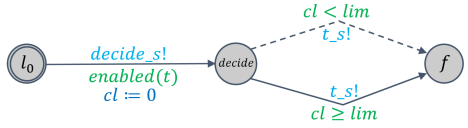

Figure 10. Modeling pattern of a TL automaton for a task with uncertain start time when the resources are occupied from the release time of task

C. The Precedence (PR) model

In the proposed framework, two types of precedence constraints have been identified. The first type, considers precedence constraints for delay between start times of tasks and is denoted as delay precedence constraint. This delay might be equal to the duration of tasks. The second type is a particular case of the first type. It considers precedence constraints for start times of tasks and is denoted as untimed precedence constraint. The automaton representing the first and second type of constraint are named delay and untimed PR automaton respectively.

Uncertainties of duration or start time of tasks does not affect the model of PR automaton. Indeed, in both types of PR automata, only the starting moment of a task is taken into account. Thus, it is not important if a task starts right after the prior task or it starts some moments later. The same goes for the duration of the task, i.e. the finishing instant of the task is not important. Precedence constraints solely set the *minimum* time distances between the starting times of tasks.

Failures which happen in idle time of resources don't cancel tasks. For that reason, by integrating this kind of failure in the problem, the model of PR automaton will not be changed.

Precedence constraints can be either untimed or delayed. The following presents the models of untimed and delay PR automata models with and without possibility of *failure while execution of task.*

1) The Untimed PR automaton model

The untimed PR model is composed of triggering transitions so as to request starting of tasks in a specific order. In Fig. 11, task t_i will be started before task t_j.

It is intended to put constraints on tasks that are fulfilled successfully. When a task fails, it will not be necessarily re-executed right after reparation of the broken-down resource. It may be executed afterwards. Therefore, its re-execution should be considered in the PR automaton model to prevent execution of tasks in an undesired order.

Figure 11. Modeling pattern of an untimed PR automata with reliable resources

To model this feature in untimed PR automata models, it is enough to add uncontrollable transitions that lead the automaton from the locations that tasks are launched to their previous locations where tasks are not launched yet. For example, when a TL launches task t_j, the PR automata in Fig. 12 takes a transition from l_1 to l_2. If t_j fails, the automaton should execute it again. To this aim, a new uncontrollable transition could be added from l_2 to l_1. Thereby, when a failure happens, a signal $stop\_s_j$ is received from the related ME automaton and the PR automata moves to the previous location. This makes it possible to execute t_j for another time. The same goes for all the tasks in the automaton. Thus, in order to consider failure during execution of tasks, the model of untimed PR automata changes to Fig. 12.

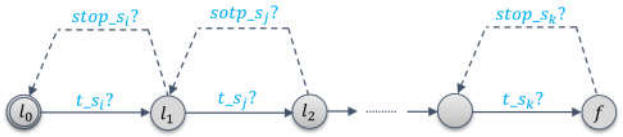

Figure 12. Modeling pattern of an untimed PR automata with tasks subject to failure

2) Delay precedence automaton

In addition to the precedence between tasks, the delay PR automaton model should demonstrate time elapse between start times of tasks. Therefore, new locations, time guards and invariants are added to the model of untimed PR automaton to

obtain delay PR automaton (Fig. 13). In this model, two locations are assigned to each task (e.g. t_j). The first location (e.g. l_2) is where the automaton waits for receiving signal t_s from the TL automaton of task t to launch the task (launching location). The second location (e.g. l_3) corresponds where it waits del time units (delay location). By receiving signal t_s, clock is reset and the automaton reaches the second location (e.g. l_3). After del time units, task t finishes and the automaton changes the location in order to wait for receiving launching signal from TL automata of the next task. Since there exist no task after execution of the last task of the constraint, naturally no delay will be defined after its execution. Thence, no delay location or guard is needed for executing the last task in this automaton. If the precedence is correctly followed, automaton will reach location f.

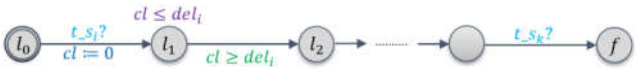

Figure 13. Modeling pattern of a delay PR automata with reliable resources

The process of modeling task failure in this automaton is similar to the untimed PR automata. The only difference is that in this model, there should exist transitions to lead the automaton from launching and delay locations to their previous launching locations. Fig. 14 represents modeling pattern of a delay PR automata with tasks subject to failure. After launching task t_i, the automaton reaches the delay location l_1. During execution of

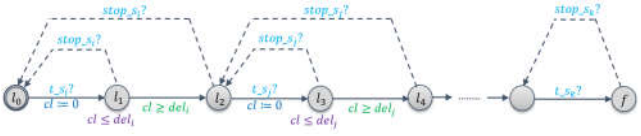

Figure 14. Modeling pattern of a delay PR automata with tasks subject to failure

the task, a resource failure may happen. There is a possibility that this failure occurs during the waiting period in l_1. Whereas, if del_i was smaller than duration of the task $d(t_i)$, this failure may happen after time del_i and when the automaton has moved to the location l_2. Thus, two uncontrollable transitions are added to the automaton. Thereby, whenever a failure signal $stop\_s_i$ is received from a ME automaton, if the PR automata was at location l_1 or l_2, it will go back to the previous launching location, i.e. initial location. Related changes for the other tasks are the same as t_i.

V. Solving Approach

In order to find an optimized schedule, the tool TIGA is used as a synthesis tool to explore the state space for determining if there is a winning strategy to reach a location where all the tasks are done and the precedence constraint is respected. For this purpose, a time-optimal control property is verified by solving a reachability game. This game concerns reachability of all the ME automata models to their initial locations and all TL and PR automata models to their final locations (f) despite uncontrollable actions of the environment. In TCTL language, this property can be formalized as follows [22]:

$$control\_t^*(u,g): A \Diamond ((\Lambda_{1\le j\le M} ME_j.l_0) \wedge (\Lambda_{1\le i\le N} TL_i.f) \wedge (\Lambda_{1\le k\le P} PR_k.f)) \quad (3)$$

where l_0 is the initial location in each ME automata, and f is the final location in each TL and PR automata. As highlighted previously, there exist M ME automata, N TL automata and P PR automata. Generally, $control\_t^*(u,g): A \Diamond goal$ means that the supervisor must reach a set of goal locations within less than $u - g$ time units. In the case of the proposed scheduling problem, goal locations are initial locations l_0 in ME automata and f locations in TL and PR automata. Moreover, u can be set to a very large number and g can have zero value. By performing a reachability game through verifying this property, TIGA yields an optimal or sub-optimal value of makespan that can be acquired despite the worst actions of the environment (from the point of view of the supervisor). Furthermore, the strategy to reach the final location can be obtained through this synthesis [22].

In the following, through an example, the difference of the optimum schedule with and without presence of uncontrollable behaviors is explained.

VI. Example

Assume there is a set of tasks $T = \{a, b, c\}$ to be done. A set of resources $R = \{R1, \dots, R5\}$ is assigned to tasks with resource association details shown in Table 1. Hence, tasks a and b are in conflict with each other, while a and c are not. Therefore, a and c can be performed simultaneously. Duration of a, b and c are 7, 5 and 3 time units respectively and the precedence constraint among tasks is such that task a should be started after finishing task b.

Table 1. Resource assignment details

task	resource				
	R1	R2	R3	R4	R5
a	✓				✓
b	✓	✓	✓		
c		✓	✓	✓	

An optimal schedule is to first execute task b and after finishing b, tasks a and c can be started simultaneously. Therefore, the minimum makespan for performing all the tasks is 12 time units.

Now let's integrate uncontrollable behaviors in the problem. Assume that duration of a is restricted to be bounded in interval $[7,11]$. In addition, resources are not reliable and may fail in their idle time or even during execution of tasks. Maximum reparation time of resources are as Table 2. Thus, the maximum duration for repairing resources necessary to execute the set of tasks $\{a, b\}$ and $\{b, c\}$ are 4 and 5 time units respectively.

The optimum makespan despite the best play of the environment is obtained 41 time units through Tiga. Fig. 15 demonstrates the Gantt chart of its corresponding schedule. It means that failures are happened at moments that can extend duration of the schedule as much as possible. Moreover, duration of task a is maximum. In this schedule, the controller executes task b at time 0. Therefore, b is executed on both sets of resources. Some instances before b ends, a failure happens in the second set of resources. This issue stops b. Maximum reparation time of the second set of resources is 5 time units. Thus, till time 10, resources are repaired and in this moment, task b restarts and

finished at time 15. Then task a and c start simultaneously. At time 18 c finished, while a is still executing.

Table 2. Repairing duration of resources

R1	R2	R3	R4	R5
4	3	4	5	4

Duration of task a is non-deterministic and is bounded in interval $[7,11]$. Until moments before time 26, the environment does not take any action to finish the task. While some instants before time 26, a failure in the first set of resources occurs which stops execution of the task. It takes 4 time units to repair the resources. At time 30, the first set of resources are repaired. Since task a was not finished, it should be repeated again. Hence, a restarts and takes 11 time units to be finished. Therefore all tasks are finished at time 41.

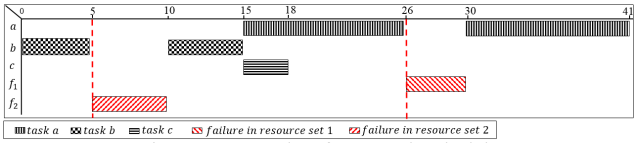

Figure 15. Gantt chart for a sample schedule

As mentioned above, this case is the worst case of uncontrollable behaviors which should be predicated for scheduling the tasks of time-critical systems. A usual schedule when the environment doesn't do its best play might be faster; and of course the best case is when no uncontrollable behavior happens.

VII. Conclusion

In this paper, a novel approach is presented to model and solve multi-resource sharing scheduling problem considering uncontrollable behaviors happening in the real life.

These kinds of uncontrollability consist of uncertain duration of tasks, uncertain start time of tasks and resource failures during a scheduled cycle. Tasks that are subject to uncertain start time are divided to two sets: those that engage resources from their decided start time and the ones, which occupy resources only when they are executing. Furthermore, two types of failures are studied: failures while tasks are being executed and failures during idle time of resources. It should be noted that all the mentioned kinds of uncontrollability are possible to be integrated in the model of the scheduling problem at the same time.

The problem is modeled by timed game automata which is a visual and expressive means of modeling. Solving a problem that contains uncontrollability needs to perform a supervisory control and extracting a strategy to reach the desired condition despite all actions of the environment. In order to solve the MRS scheduling problem, time-optimal reachability game, which is a kind of supervisory control, is performed through the synthesis tool Tiga.

As future research, other kinds of uncontrollability such as uncontrollable arrival of tasks during the schedule, malfunction or unavailability of resources can be integrated in the model.

References

[1] E. B. Edis, C. Oguz, and I. Ozkarahan, "Parallel machine scheduling with additional resources: Notation, classification, models and solution methods," 2013.

[2] M. Afzalirad and J. Rezaeian, "Resource-constrained unrelated parallel machine scheduling problem with sequence dependent setup times, precedence constraints and machine eligibility restrictions," *Comput. Ind. Eng.*, vol. 98, pp. 40–52, 2016.

[3] K. R. Quintero Garcia, "Optimisation d'alignements d'un réseau de pipelines basée sur les algèbres tropicales et les approches génétiques," Lyon, INSA, 2015.

[4] Ibm, "IBM ILOG CPLEX Optimization Studio," *Man. Seq. interval Var.*, p. 594, 2014.

[5] Y. Abdeddaïm, E. Asarin, and O. Maler, "Scheduling with timed automata," in *Theoretical Computer Science*, 2006, vol. 354, no. 2, pp. 272–300.

[6] G. Behrmann, E. Brinksma, M. Hendriks, and A. Mader, "Production Scheduling by Reachability Analysis - A Case Study," in *19th IEEE International Parallel and Distributed Processing Symposium*, 2005, p. 140a–140a.

[7] S. Subbiah and S. Engell, "Short-term scheduling of multi-product batch plants with sequence-dependent changeovers using timed automata models," *Comput. Aided Chem. Eng.*, vol. 28, no. C, pp. 1201–1206, 2010.

[8] A. R. Shehabinia, L. Lin, and R. Su, "Timed Supervisory Control for Operational Planning and Scheduling under Multiple Job Deadlines," *arXiv Prepr. arXiv1607.04255*, 2016.

[9] S. Panek, O. Stursberg, and S. Engell, "Efficient synthesis of production schedules by optimization of timed automata," *Control Eng. Pract.*, vol. 14, no. 10, pp. 1183–1197, 2006.

[10] P. Marange, J.-F. Petin, A. Manceaux, and D. Gouyon, "Contribution à la reconfiguration des systèmes de production : ordonnancement par recherche d'atteignabilité," *J. Eur. des Systèmes Autom.*, 2011.

[11] S. Gaubert and J. Mairesse, "Task Resource Models and (max,+) Automata," vol. 11, pp. 133–144, 1995.

[12] G. Rzevski and P. Skobelev, "Intelligent Adaptive Schedulers For Railways," *Int. J. Transp. Dev. Integr.*, vol. 1, no. 3, pp. 414–420, Apr. 2017.

[13] U. Dorndorf, F. Jaehn, and E. Pesch, "Flight gate assignment and recovery strategies with stochastic arrival and departure times," *OR Spectr.*, vol. 39, no. 1, pp. 65–93, 2017.

[14] N. Kundakcı and O. Kulak, "Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problem," *Comput. Ind. Eng.*, vol. 96, pp. 31–51, 2016.

[15] A. Cimatti, A. Micheli, and M. Roveri, "Strong Temporal Planning with Uncontrollable Durations : A State-Space Approach," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015, pp. 3254–3260.

[16] A. David, J. Illum, K. G. Larsen, and A. Skou, "Model-based Framework for Schedulability Analysis Using Uppaal 4.1," in *Model-Based Design for Embedded Systems*, 2009, pp. 1–32.

[17] E. Dumitrescu, A. Girault, H. Marchand, and E. Rutten, "Multicriteria optimal reconfiguration of fault-tolerant real-time tasks," in *IFAC Proceedings Volumes (IFAC-PapersOnline)*, 2010, vol. 10, no. PART 1, pp. 356–363.

[18] R. Su, J. H. Van Schuppen, and J. E. Rooda, "The synthesis of time optimal supervisors by using heaps-of-pieces," *IEEE Trans. Automat. Contr.*, vol. 57, no. 1, pp. 105–118, 2012.

[19] R. Boukra, S. Lahaye, and J.-L. Boimond, "New representations for (max,+) automata with applications to performance evaluation and control of discrete event systems," *Discret. Event Dyn. Syst.*, vol. 25, no. 1–2, pp. 295–322, 2015.

[20] F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime, "Efficient on-the-fly algorithms for the analysis of timed games," in *CONCUR: International Conference on Concurrency Theory*, 2005, vol. 5, pp. 66–80.

[21] R. De Munter, "A comparison of Timed Games and Time Optimal Supervisor Synthesis," 2010.

[22] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime, "Uppaal Tiga User-manual." Aalborg University, 2007.