Using UPPAAL for the secure and optimal control of AGV fleets

Yoann Arnaud, Jean-Louis Boimond, Jose E.R. Cury, J.J. Loiseau, Claude Martinez

Using Uppaal for the secure and optimal control of AGV fleets

Yoann Arnaud * **Jean-Louis Boimond** ** **José E.R. Cury** ***
Jean Jacques Loiseau * **Claude Martinez** *

* *IRCCyN, 1 rue de la Noë, BP 92101, 44321 Nantes CEDEX 03, France (e-mail: {yoann.arnaud,jean-jacques.loiseau,claude.martinez}@irccyn.ec-nantes.fr)*
** *LISA, Université d'Angers, ISTIA, 62 Avenue Notre Dame du Lac, 49000 Angers (e-mail: jean-louis.boimond@istia.univ-angers.fr)*
*** *LCMI-DAS, Universidade Federal de Santa Catarina, Florianópolis SC 88040-900, Brazil (e-mail: cury@das.ufsc.br)*

Abstract: The design and realization of an on line control system for automated guided vehicles (AGV) is addressed. A synthesis method is proposed based on the use of the model checking tool for timed automata Uppaal. This system has to route the vehicles while ensuring the system safeness, a good coordination between vehicles and the optimization of performance criteria. This problem is like synthesizing a Ramadge and Wonham supervisor combined with routing and optimizing functions, that is an ongoing problem within the supervisory control theory. The proposed concepts are validated through a software tool suite based on Uppaal in order to generate optimal traces and interact with an AGV system emulated with Arena.

Keywords: Timed state automata,Uppaal, Control, Optimization, Automated guided vehicules.

1. INTRODUCTION

In this paper, a synthesis method for the design of an automated guided vehicle (AGV) fleet control is proposed. This control is based on the use of formal checking on timed automata. This control system routes the vehicles along a circuit in a production workshop in order to accomplish missions allocated to the system. It must ensure that no conflict or deadlock between vehicles occurs. Moreover, the routing is calculated on-line so as to minimize the total execution time of the missions allocated.

Methods from the Operational research domain have been proposed ([Reveliotis (2000)], [Lawley *et al.* (1997)]) for designing AGV fleet control systems. These methods are based on optimization techniques in order to calculate a good planification. These kinds of techniques are open-loop and do not take into account drifts or breakdowns. To solve this problem, S. Maza ([Maza and Castagna (2005)]) proposed to merge predictive scheduling techniques to on-line reroutings that ensure the conservation of a deadlock-free behaviour. The global optimization is lost and, a priori, some possibilities are forgotten, to simplify the planification. It is said that the behaviour is not maximaly permissive.

Another solution for designing AGV fleet controls is using the Ramadge and Wonham supervision control theory ([Ramadge and Wonham (1987)], [Ramadge and Wonham (1988)], [Cassandras and Lafortune (1999)]) for solving conflict and deadlock problems, linked with optimization functions for calculating an optimal route. Thanks to this theory, the existence of a solution is proved and the best solution is conserved (maximal permissivity).

Combining supervision and optimization has been realised, by adding costs on edges of automata that modelize AGV systems. General theories have been developped by some authors ([Grigorov and Rudie (2006)], [Kumar and Garg (1995)], [Sengupta and Lafortune (1998)]) and inspired precedent works of Arnaud *et al.* ([Arnaud *et al.* (2009)]) where an energy criterion is minimized. However, due to parallelism between the AGV, a time criterion cannot be optimized by adding costs on automata and therefore no algorithm has been proposed in this direction.

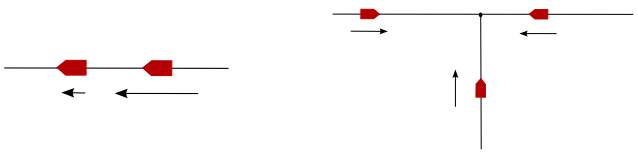
We propose a new method based on the modelization of AGV fleet systems with timed automata. The existence of a control law satisfying the specifications of deadlock and conflict avoidance and the calculation of a route planning is like verifying the existence of a particular trace in a timed automaton and finding this trace. The proposed method is efficient for the synthesis of an AGV fleet control. Checking our model with timed automata garanties maximal permissivity, deadlock and conflict avoidance and a small calculation time. Moreover, optimizing the total time to accomplish missions is possible. Effective verification and synthesis are realized with the help of Uppaal ([Bengtsson *et al.* (1996)]), a software tool that implements the formal verification algorithms for timed automata.

An AGV system is composed of a set of vehicles V_i dedicated to the realization of tasks. The vehicles travel along a circuit, divided in sections and intersections. Sections are labelled with upper case letters. An AGV moving on

section DA travels on the section linking intersections A and D, from D to A. The tasks to be accomplished are called missions and consist in picking up and delivering manufactured parts on particular intersections called workstations and denoted W_i.

The AGV control system has to ensure four functions. The first one consists in routing on-line each vehicule on the circuit in order to accomplish the missions allocated by an upper level control system. The second function consists in avoiding conflicts between vehicles (figure 1.1) as the occupation by several AGV of a same section. This ensures security for the process. The third function ensures a good coordination of AGV on the circuit so that no deadlock situation occurs (figure 1.2). Finally, the fourth one concerns the selection of an optimal route that minimizes the overall time needed by the AGV to accomplish the missions allocated. Note that generation of missions and their dispatching are not addressed here.

The second and third functions of the control system, the conflict and deadlock avoidance, are usually solved by the supervisory control theory. Therefore a solution to the design problems could be reformulated in terms of minimal time routing in a supervised automaton. As we shall see, this argument permits to assess the existence of a solution to the design problem, under mild assumptions. However, as mentionned above, this approach does not lead to an effective method. We shall not use this theory in practice. The problem can be reformulated in terms of verification of a timed automaton and effectively solved using software tools that implement the algorithms from formal model checking.



1.1 Conflict 1.2 Deadlock

Fig. 1. Situations to be avoided

The study of this article is about the class of AGV systems composed of a garage, several workstations, a bidirectional circuit and monodirectional vehicles.

The structure of this article is as follows. After this introduction, we present in section 2 some basic tools about timed automata, the tool Uppaal, and a formal proof for the existence of a solution of our control design problem. In section 3, the general techniques for modelizing and solving our control problem with Uppaal are detailed. Section 4 presents the works of implementation of these techniques in a software tool suite in order to validate the concepts on an example. Finally, this article is concluded by recalling the main contribution of this work in the framework of AGV fleet control.

2. BASIC TOOLS

2.1 Timed Automata

This subsection is based on the work of Patricia Bouyer presented in [Bouyer (2005)]. Let $\mathbb{T}$ be the time axis, often assumed to be the set of non negative rational numbers $\mathbb{Q}_+$ or equivalently, chosen as the set of natural numbers $\mathbb{N}$.

A timed automaton over $\mathbb{T}$ is a tuple (Σ, Q, T, I, F, X) where:

- Σ is an alphabet of actions of finite dimension,
- Q is a finite set of states,
- X is a finite set of clocks,
- $I \subseteq Q$ is a set of initial states,
- $F \subseteq Q$ is a set of final states,
- $T \subseteq Q \times [C(X) \times \Sigma \times 2^X] \times Q$ is a finite set of transitions where $C(X)$ is a set of constraints over clocks described below. An element of T is denoted $q \xrightarrow{g,a,Y:=0} q'$.

Let X be a set of clocks, $C(X)$ is a set of constraints over clocks, over clock differences and over conjunctions of previous constraints. $C(X)$ is defined as follows:

$$g ::= x \bowtie c \mid x - y \bowtie c \mid g \wedge g \mid true,$$

where $x \in X$, $y \in X$, $c \in \mathbb{Z}$, and $\bowtie \in \{<, \leq, =, \geq, >\}$.

In the previous definition, c is chosen in $\mathbb{Z}$. This choice does not imply loss of generality because a timed automaton with clock constraints defined over $\mathbb{Q}$ can be equivalently reduced to a timed automaton with clock constraints defined over $\mathbb{Z}$. Let v a function returning a clock valuation, one say that v satisfies a constraint g ($v \vDash g$), g being of type $x \bowtie c$ (resp. $x - y \bowtie c$), when $v(x) \bowtie c$ (resp. $v(x) - v(y) \bowtie c$).

Let a set $Y \subseteq X$, one denote $[Y \leftarrow 0]v$ the value such that, $\forall\, x \in Y, ([Y \leftarrow 0])v(x) = 0$ and $\forall\, x \in X \backslash Y, ([Y \leftarrow 0])v(x) = v(y)$.

Let a time sequence τ be a finite non decreasing sequence built over elements of $\mathbb{T}$ ($\tau = (t_i)_{1 \leq i \leq p} \in \mathbb{T}^*$). A word σ (or a trace) is a sequence of actions $(a_i)_{1 \leq i \leq p} \in \Sigma^*$. Finally, a timed word, also called a timed trace, w is a sequence $(a_i; t_i)_{1 \leq i \leq p} \in (\Sigma \times \mathbb{T})^*$. Such a timed word is said to be accepted by the considered automaton if it corresponds to the execution trace of the automaton.

$$(q_0, v_0) \xrightarrow[t_1]{g_1,a_1,Y_1} (q_1, v_1) \quad \dots \quad \xrightarrow[t_p]{g_p,a_p,Y_p} (q_p, v_p),$$

$$\begin{cases} v_0(x) = 0, \forall x \in X, \\ v_{i-1} + (t_i - t_{i-1}) \vDash g_i, \\ v_i = [C_i \leftarrow 0](v_{i-1} + (t_i - t_{i-1})). \end{cases}$$

A timed automaton A_t admits a timed language $L_t(A_t)$ which is defined as the set of all words accepted by A_t. One define the untimed language $Untime(L_t(A_t))$ as a projection of language $L_t(A_t)$, such that:

$$Untime((a_i, t_i)_{1 \leq i \leq n}) = (a_i)_{1 \leq i \leq n},\ n \in \mathbb{N}, \quad \forall (a_i, t_i)_{1 \leq i \leq n} \in L_t(A_t).$$

The number of configurations (pairs (q, v) composed by a state and a clock valuation) is infinite for timed automata because of the clock valuation v. Hence the direct transposition of verification methods on classical automata is not possible. Nevertheless, Alur and Dill have developed an abstraction technique in [Alur and Dill (1994)]. Thanks to this technique a timed automaton can be represented as a region graph which is a finite automaton. So, the reachability problem in a timed automata is equivalent to

a reachability problem in a finite automaton and therefore is decidable.

2.2 Modeling timed systems with Uppaal

Uppaal is a tool for modeling timed discrete event systems and verifying some of their properties [Bengtsson *et al.* (1996)]. We briefly recall the use of this tool in this section.

Modeling:

The following data types may be used to define data in an Uppaal model:

- *bool*: boolean values,
- *int*: integer values,
- *clock*: clocks,
- *chan*: objects used to synchronize events between *templates*,
- *struct*: data composed of data types cited above.

Types listed above may be declared in environments *declaration* of the Uppaal model. Functions that use classical patterns like *for*, *while*, *if*, may also be declared in the same environments.

A *template* represents a subset of the global system that is modeled as a graph composed of *locations* and *edges*. It contains a declaration section where a local clock may be defined. The *locations* are the nodes of a *template* graph, they are identified by their names. An *invariant* can be assigned to a *location*, this is a boolean expression involving variables and clock valuations that forbid or not to access a node. Furthermore, instanciation of *templates* is possible, which might be usefull to model systems made of identical subsystems.

The following features may be assigned to an *edge*:

- *Selection*: a selection allows to define a range of values that may take a variable in the model.
- *Guard*: a guard is a boolean condition that enable the firing of an *edge*. A guard may involve clock values and variables of the model.
- *Synchronisation*: when *edges* of different *templates* have to be synchronized, one assign a channel object of type *chan* to these *edges*.
- *Update*: this feature is used for updating the variables and clocks after the firing of an *edge*.

We recall that the model defined in Uppaal is not explicitly a timed automaton, but it is important to notice that the semantic is exactly equivalent. It is clear that the concept of *guard* corresponds to constraints defined in section 2.1. The remaining data types, i.e. templates, boolean variables, and invariants, corresponds to automaton with guards. The notion of automata products explains the use of templates to define timed automata. The use of boolean variables is equivalent to the manipulation of state automata. Finally, invariants are equivalent to guards.

Model checking:

Reachability problems are decidable (section 2.1) for the class of timed automata considered. So, some properties of timed automata can be verified with Uppaal, if they are formulated as reachability problems. In Uppaal, a property should be formulated using TCTL, a common useful language for expressing properties. *Queries* are then like "$\Box$ *expression*" where $\Box \in \{E[\,], E<>, A[\,], A<>\}$, which means:

- E $<>$: Does exist a trace leading to a state that satisfy *expression*?
- $E[\,]$: Does exist a trace, finite or not, such that all states satisfy *expression*?
- A $<>$: Do all traces lead to a state such that *expression* is satisfied?
- $A[\,]$: Do all states satisfy *expression*?

The region graph is explored by Uppaal like a tree using *branch and bound* algorithms. Exploration options, *breadth-first* or *depth-first* are proposed. If the checker's answer to a property is true, a *diagnotic trace* can be returned. *Some* trace corresponds to the first solution found. The *shortest* one corresponds to the trace that involve a minimum number of state evolution. Finally, the *fastest* corresponds to a minimum execution time.

The choice of the exploration option implies different behaviours of the checker, and may impact its efficiency. Exploring deeply first will be more expensive in terms of processor time consumption and less expensive in terms of memory consumption than exploring widely first. This choice has to be made depending on the behavior of the checker with respect to a given model. Concerning the *diagnostic trace*, the behaviour of the algorithm will depend on the following options. If *some* trace is chosen, the algorithm will stop after it finds a trace that satisfies the query, if one exists. If an optimal trace is expected (*shortest* or *fastest*), then the region graph will be totally explored in order to find the best solution among the existing ones. Obviously, this takes more time than finding *some* trace.

2.3 Existence of a solution

This subsection is devoted to prove the existence of a solution to the problem of designing a control that routes the AGV on the circuit, prevents conflicts between AGV, avoids deadlock situations and optimizes the route to be taken by the vehicles.

This problem can be seen as a Ramadge and Wonham supervision problem combined with an optimization problem. The second and third functions, about preventing conflicts and deadlocks, can be solved with the use of the supervision control theory of Ramadge and Wonham. This theory is based on the use of automata for modeling a system called G. Events are defined on edges, they can be controlable or not. The theory consists in generating a supervisor A which will act on controlable events of G to forbid their occurence, if they lead the system to a situation that does not respect some specifications. In the case of AGV system study, all the events in G are controlable (they all can be forbidden if necessary) and the specifications consist in forbidding conflicts situations. The result for this particular case is that the supervisor A exists and is a sub-automaton of G ($A \subseteq G$). The states in G that are not in A are states that generate conflicts or deadlocks. Moreover the automaton is coaccessible with respect to an initial state for the process. This gives the

property of deadlock avoidance. We make the hypothesis that the circuit is well designed, so that the AGV can reach any point of the circuit, and then come back to its garage. So, we can assert that at least one solution that prevents conflicts and dealocks exists in A. For instance, one solution consists in moving each AGV one after another in order to achieve missions one by one.

Let us now define the automaton A_t based on A, and temporized with clock constraints. As temporizing an automaton reduces its accessible state space, the untimed language generated by A_t is included in the language generated by A ($Untime(L_t(A_t)) \subseteq L(A)$). If the set of clock constraints is reduced to constraints like $x \geq c$ ($x \in X,\ c \in \mathbb{N}$), the accessible state space is not reduced since every configuration for the AGV system in A is accessible in A_t too if the time elapsed in A_t is high enough. This implies the following statement :

$$Untime(L_t(A_t)) = L(A),\ \text{if } x \geq c,\ \forall x \in X,\ \forall c \in \mathbb{N}$$

Therefore, the existence of a solution is proved, provided the timed constraints are as above. This justifies the use of UPPAAL to calculate a solution.

3. SYNTHESIS OF AN AGV CONTROL SYSTEM USING UPPAAL

Recall the problem we are trying to solve. Let a mission, at most, provided for each AGV at a time. We are looking for a sequence of movements to be send to the vehicles to ensure that all the missions are achieved as quickly as possible. In this section, we present the general techniques for modeling an AGV system with UPPAAL and solving this control problem, so that the four following functions are respected: first, routing the vehicles, second, managing conflicts, third, preventing the occurrence of deadlock and fourth, selecting an optimal route. For the sake of our analysis, these various problems and their solutions are exposed separately in this section. However, at the level of the implementation, they are solved simultaneously during the checking procedure.

Designing the UPPAAL model is achieved with two steps. First a basic model representing the movement of vehicles without any restriction is designed (section 3.1). Then we present in sections 3.2, 3.3 and 3.4 how to complete the modelisation in order to fulfill the four functions cited above.

3.1 Basic model

The basic model has to represent the movement of vehicles on the studied circuit. For this purpose, the logical behaviour of each AGV is represented in a *template*. Each time an AGV moves on a new section, the garage or workstations, a new location in the template is set, which reflects a change of the model's state. For example, if the location $W2$ is set, then the AGV occupies the workstation W_2. If a location DA is set, the AGV is moving on the section between A and D from D to A. For practical reasons, each location in a template is named like XX_xx where XX is the name of the section and xx its number. All the AGV templates are similar (same locations and edges), except for the location named $Init$ that can be linked to any location depending on the AGV positions on the circuit. Practically, this location $Init$ allows a dynamic definition of the AGV's actual position on the circuit for each template.

In order to keep a track of the position of each AGV on the circuit, we define a variable vector *pos* of size n_{agv}, where n_{agv} is the number of AGV composing the fleet. $pos[i]$ stores the number of the section or workstation occupied by the AGV V_i and is updated by adding the appropriated statement $pos[i] = xx$ in the *update* field of each edge.

3.2 Realization of missions

The first function is about routing the vehicles on the circuit, so that the missions allocated to the AGV are achieved. With UPPAAL, this consists in finding a particular trace in the timed automata represented by the model, that corresponds to the achievement of these missions. The completion of missions is stored in the logical variable $AGVOK$. So, this problem consists in finding a trace in the timed automaton that leads to a state verifying $Missions\_OK == True$ where

$$Missions\_OK = \prod_{i=1}^{n_{agv}} AGVOK[i]$$

and where $AGVOK[i]$ is true when the AGV V_i has accomplished its mission. In the model, the *update* field on edges updates each $AGVOK[i]$ as soon as an AGV arrives to the workstation corresponding to its mission.

3.3 Conflict and deadlock avoidance

The second function is about forbidden conflictive situations, that means the simultaneous occupation of a workstation or a section by at least two AGV. This ensures the security of the process. This problem consists in locking the firing of some edges, if this firing make the template access to particular locations. This is realized with guards on edges where a function $is\_free()$ is added. This function returns a boolean value *true* (resp. *false*) if the access to a section shall be authorized (resp. forbidden).

The third function is about preventing the occurence of deadlocks. This ensures a good coordination between all the AGV on the circuit. This problem can be formulated as a coaccessibility problem. If a state of the timed automaton is coaccessible with respect to the initial state of the process (in our case, all the AGV in the garage, location number n_0), so we can say that being in this state will not generate a deadlock situation. This property comes from results from the supervision control theory (section 2.3). Therefore, the problem consists in verifying that the initial state of the process is accessible, once the missions have been achieved. This is a similar problem to the one of achieving missions and can be solved with the use of logical variables. We represent the initial situation of the process with the help of a boolean variable $Home$ where

$$Home = \prod_{i=1}^{n_{agv}} (pos[i] == n_0)$$

3.4 Verification and optimization

The last function consists in choosing an optimal route for the AGV that minimizes the total time required for achieving the missions allocated. Recall that UPPAAL has also been used in order to solve optimization problems. Some works are presented by Behrmann *et al.* in [Behrmann *et al.* (2005)]. Similarly, we are exploiting this tool for optimizing a time criterion. This problem is solved with UPPAAL by asking a *diagnostic trace* with the *Fastest* option, so that the total time is minimized. The following query is defined:

$$E <> Missions\_OK \ \cdot \ Home$$

That means: Is there, in the region graph, a trace representing the achievement of the missions and leading to the initial state of the process? The time parameters to be taken into account are the minimum duration of stay of vehicles on circuit's sections or workstations. So, the firing of some edges is forbidden as long as this duration is not elapsed. This is modeled with the addition of time constraints on guards. As seen in section 2.3, clock constraints have to be as

$x \geq c$, where x is a clock associated to a template and c is the minimum duration imposed,

in order to garanty the systematic existence of a solution. Moreover clocks have to be reset each time an AGV moves. So, we add $x := 0$ in the update fields of edges. At this stage, the modelisation allows us to achieve the mission and return to the initial state of the process as quickly as possible. However, only the realization of tasks in minimal time is sought. We shall therefore, in the model, reset all the clock constraints c just after the logical variable $Missions\_OK$ (that reflects the achievement of missions) becomes true. That virtually freezes the flow of time at this date and that corresponds now to our optimization goal.

4. IMPLEMENTATION AND VALIDATION ON AN EXAMPLE

The general techniques previously presented have been applied on an example and implemented in a software tool suite that controls an AGV system emulated with ARENA([Kelton *et al.* (2000)]), a tool for simulating discrete event systems.

4.1 The AGV system

This circuit of the example AGV system is represented in figure 2. The system is composed of a set of three AGV V_i, $i \in \{1, 2, 3\}$, and a circuit made of sections, four intersections (A, B, C, D), two workstations (W_1, W_2) and a garage (W_0).

This AGV system seems quite simple but is actually complex due to the fact that all AGV can access to all sections in both directions. Indeed, a template contains 19 locations and 36 transitions, that makes 6859 (19^3) different situations and 38988 ($3 \times 19^2 \times 36$) different evolutions between all these situations.

The control of this AGV system respecting the safety, coordination and optimization constraints outlined earlier

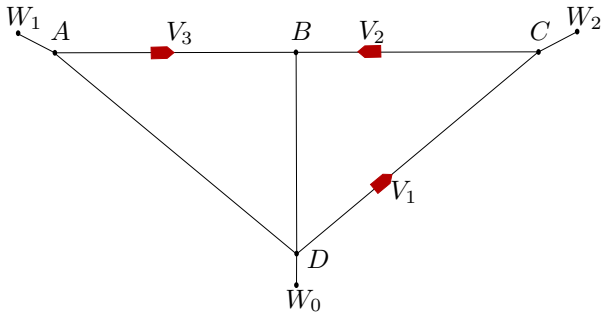


Fig. 2. The AGV system studied

is implemented with UPPAAL according to the principles explained in section 3.

The modelisation of this AGV system leads to AGV's templates of the following form (figure 3):

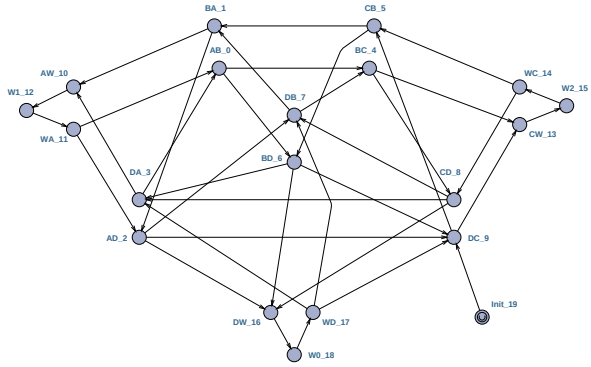


Fig. 3. One AGV's template.

The exploration of the solutions is achieved with the search order *breadth-first* which provides for this model a better response delay, as shown by the comparisons that have been conducted experimentally on randomly selected cases (cf. table below). Moreover, the memory consumption is too small to be measurable.

Initial state	Missions	breadth (s)	depth (s)	ratio
(12,5,18)	(-,15,12)	1,22	21,23	17,4
(18,2,7)	(15,-,12)	0,81	7,15	8,8
(6,1,4)	(12,15,-)	1,31	89,11	68,0
(11,18,9)	(15,15,12)	3,63	38,97	10,7
(2,9,6)	(15,15,12)	2,31	23,43	10,1
(3,8,15)	(15,12,12)	3,02	85,35	28,3

4.2 Real-time implementation and validation

The AGV control system previously presented has been implemented through a software tool suite. A similar procedure has already been introduced in [Arnaud *et al.* (2009)].

The structure of the software tool suite is represented in figure 4. The AGV system is emulated by a real-time simulator, built using the simulation tool for discrete event systems ARENA. A software tool has been written for managing the calculation of optimal traces with UPPAAL and the communication with the emulator. The missions to be achieved are specified by the user thanks to the graphical user interface. The UPPAAL model *xml* file is

generated in real time, actually every time a new mission is defined. The *xml* file and the verification query are then analysed by the executable *verifyta.exe* which comes with the software tool UPPAAL. An optimal trace is then generated. It defines how to schedule orders to be sent to AGV.

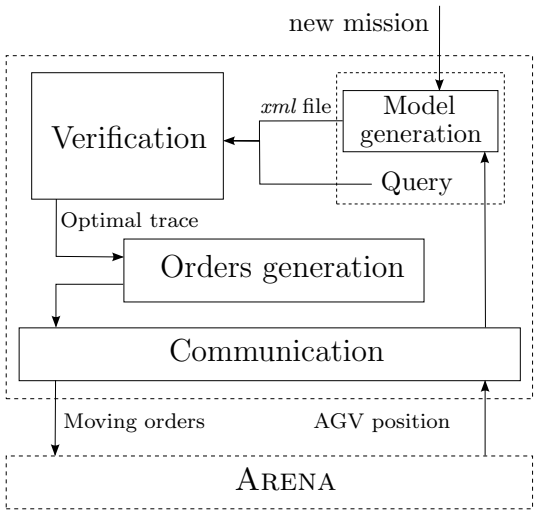


Fig. 4. Structure of the software tool suite.

5. CONCLUSION

A new method for the design of a control system for an AGV fleet is presented. The first function of the control system is to route the vehicles through their circuit, to realize the missions that are allocated on line to the AGV system. The different vehicles are routed in parallel, which generates risky situations. Conflict (accident) and deadlock avoidance are hence two important functions of the AGV system. Finally, the minimization of the time required to achieve the missions is a fourth objective of the control system. The existence of a solution to this design problem is shown. Formally, this existence is a consequence of the supervisory control theory. The solution could be obtained by the superposition of a supervisor coming from this theory, and an optimization algorithm. Practically, no such algorithm has been described and this approach, leads to an open problem. An alternative is developed, that is based on the use of the theory of timed automata. The design problem is reformulated in terms of verification of a timed automaton, that proceeds from the description of the AGV system and of the control system specifications. UPPAAL, a powerful tool that implements the verification algorithms for timed automata, is used to calculate a solution to the control design problem. A software tool suite has been developed, that permits the implementation of the proposed design method. The approach is validated on a realistic example.

REFERENCES

[Alur and Dill (1994)] R. Alur, D. Dill. *A theory of timed automata*. Theoretical Computer Science, volume 126:2, pages 183-235, 1994.

[Arnaud *et al.* (2009)] Y. Arnaud, J.E.R. Cury, J.J. Loiseau and C. Martinez. *Pilotage sûr et optimal d'une flotte de véhicules autoguidés*. Journées Doctorales MACS, Mars 2009.

[Behrmann *et al.* (2005)] G. Behrmann, K. Larsen, J. Rasmussen. *Optimal scheduling using priced timed automata*. ACM SIGMETRICS Performance Evaluation Review, volume 32:4, pages 34-40, ACM Press, 2005.

[Bengtsson *et al.* (1996)] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson and W. Yi. UPPAAL*: a tool suite for automatic verification of real-time systems*. Lecture Notes in Computer Science, Hybrid Systems III, volume 1066/1996, pages 232-243, 1996.

[Bouyer (2005)] P. Bouyer. *An introduction to timed automata*. École d'été Temps Réel 2005, Nancy, Septembre 2005.

[Cassandras and Lafortune (1999)] C.G. Cassandras and S. Lafortune. *Introduction to discrete event systems*. Kluwer Academic Publishers, 1999.

[Grigorov and Rudie (2006)] L. Grigorov and K. Rudie. *Near-optimal online control of dynamic discrete-event systems*. Discrete Event Dynamic Systems, volume 16:4, pages 419-449, 2006.

[Kelton *et al.* (2000)] D. D. Kelton and D. A. Sadowski and R. P. Sadowski. *Simulation with Arena*. McGraw-Hill School Education Group, 2000.

[Kumar and Garg (1995)] R. Kumar and V. Garg. *Optimal supervisory control of discrete event dynamical systems*. SIAM Journal on Control and Optimization, volume 33:2, pages 419-439, 1995.

[Lawley *et al.* (1997)] M. Lawley, S. Reveliotis and P. Ferreira. *Design Guidelines for Deadlock-Handling Strategies in Flexible Manufacturing Systems*. International Journal of Flexible Manufacturing Systems, volume 9:1, pages 5-30, 1997.

[Maza and Castagna (2005)] S. Maza and P. Castagna. *A performance-based structural policy for conflict-free routing of bi-directional automated guided vehicles*. Computers in Industry, volume 56:7, pages 719-733, 2005.

[Ramadge and Wonham (1987)] P.J. Ramadge and W.M. Wonham. *Supervisory control of a class of discrete event systems*. SIAM Journal of Control and Optimization, volume 25:1, pages 206-230, 1987.

[Ramadge and Wonham (1988)] P.J. Ramadge and W.M. Wonham. *The control of discrete event systems*. Proceedings of the IEEE, volume 77:1, pages 81-98, 1989.

[Reveliotis (2000)] S.A. Reveliotis. . IIE Transactions, volume 32, pages 647-659, 2000.

[Sengupta and Lafortune (1998)] R. Sengupta and S. Lafortune. *An optimal control theory for discrete event systems*. SIAM Journal on control and Optimization, volume 36:2, pages 488-541, 1998.