UNIVERSITÄT
KOBLENZ · LANDAU
Institut für Informatik

FB 4
Informatik

Multi-agent systems: Modeling and Verification using Hybrid Automata

Ammar Mohammed
Ulrich Furbach

Nr. 10/2009

Arbeitsberichte aus dem Fachbereich Informatik

Arbeitsberichte des Fachbereichs Informatik

ISSN (Print): 1864-0346
ISSN (Online): 1864-0850

Herausgeber / Edited by:

Der Dekan:
Prof. Dr. Zöbel

Kontaktdaten der Verfasser

Ammar Mohammed, Ulrich Furbach
Institut für Informatik
Fachbereich Informatik
Universität Koblenz-Landau
Universitätsstraße 1
D-56070 Koblenz
EMail: ammar@uni-koblenz.de, uli@uni-koblenz.de

Multi-agent systems: Modeling and Verification using Hybrid Automata

Ammar Mohammed and Ulrich Furbach

Universität Koblenz-Landau, Artificial Intelligence Research Group, D-56070 Koblenz,
{ammar,uli}@uni-koblenz.de

Abstract. Hybrid automata are used as standard means for the specification and analysis of dynamical systems. Several researches have approached them to formally specify reactive Multi-agent systems situated in a physical environment, where the agents react continuously to their environment. The specified systems, in turn, are formally checked with the help of existing hybrid automata verification tools. However, when dealing with multi-agent systems, two problems may be raised. The first problem is a state space problem raised due to the composition process, where the agents have to be parallel composed into an agent capturing all possible behaviors of the multi-agent system prior to the verification phase. The second problem concerns the expressiveness of verification tools when modeling and verifying certain behaviors. Therefore, this paper tackles these problems by showing how multi-agent systems, specified as hybrid automata, can be modeled and verified using constraint logic programming(CLP). In particular, a CLP framework is presented to show how the composition of multi-agent behaviors can be captured dynamically during the verification phase. This can relieve the state space complexity that may occur as a result of the composition process. Additionally, the expressiveness of the CLP model flexibly allows not only to model multi-agent systems, but also to check various properties by means of the reachability analysis. Experiments are promising to show the feasibility of our approach.

1 Motivation

Specifying behaviors of (physical) multi-agent systems is a sophisticated and demanding task, because of the high complexity of the interactions among agents and the dynamics of the environment. An important aspect of multi-agent systems is that the agents interact with a physical environment. Such interactions typically consist of continuous changes of behaviors of agents (e.g. a movement of a robot, or an agent is waiting for occurrence of an event), as well as discrete changes of behaviors. Those scenarios can be captured by means of hybrid automata [12]. Here the discrete changes are modeled using a form of transition diagrams dialect like statecharts [26], while the continuous changes are modeled using differential equations. Hybrid automata formal semantics make them accessible to formal validation of systems, especially for systems, which are situated in safety critical environments.Thus, it is possible to prove desirable features as well as the absence of unwanted properties for the modeled systems automatically with the help of hybrid automata verification tools [13, 8, 3].

Hybrid automata can be used to model and verify multi-agent plans (we call it modeling multi-agent systems), especially for those agents that are defined through their capability to continuously react to a physical environment, while respecting some time constraints. With the help of the versification's tools of hybrid automata, one can validate/verify and control of multi-agent plans. For this reason, several researches, for example [6, 7, 9, 22, 23], have approached hybrid automata as a framework in oder to model multi-agent systems in a dynamic environment, where the time is critical. There are authors, for example [18], who have modeled multi-agent systems with a simple form of hybrid automata that are called timed automata [2]. Nevertheless, two problems occur when applying hybrid automata to multi-agent systems. Firstly, multi-agent systems are specified as a network of synchronized hybrid automata that have to be parallel composed statically into an automaton (synonymy agent). By statically we mean that agents have to be parallel composed prior to the verification phase. Technically, the composition of hybrid automata is obtained from the cartesian product of the number of states of all concurrent automata, unless the automata have mutual synchronization messages. In this case, the states have to be considered simultaneously. As a result of the composition process, an agent captures all possible behaviors that may occur in the multi-agent systems. In turn, the resulting composed agent afterwards is checked by hybrid automata verification tools. Consequently, this composition process may lead to a state explosion problem.

The second problem concerns the expressiveness of the modeling tools. Standard hybrid automata tools are not flexible enough to model multi-agent systems. This is for the reason that they are special purpose tools, which model the agents' decision depending on the evaluation of continuous dynamics. However, there are favorable situations of modeling multi-agent systems where the agents' decision steps do not depend on the evaluation of continuous dynamics, but on evaluation functions (e.g. shortest distance, max, or min) happening during the continuous dynamic. Imagine, for example, an agent who wants to cooperate with the nearest agent to conduct certain tasks in a rescue team of a multi-agent system. To our knowledge, this type of decision making is beyond the capabilities of the current hybrid automata verification tools. Therefore it is necessary to have expressive tools that can handle such situations. Ideally, modeling tools are favorable when they are flexibly able to verify the systems' requirements.

To this end, the purpose of this paper is to cope with the mentioned problems when approaching hybrid automata to model multi-agent systems. In particular, we present a novel approach which models hybrid automata based on constraint logic programming. This approach is appropriate to represent multi-agent systems specified as hybrid automata. The novelty of the presented approach is that the composition of hybrid automata is built dynamically on the fly, where only the reached behaviors are captured dynamically, rather than building all possible behaviors in advance. On the other hand, the expressiveness of CLP does not only allow us to model multi-agent systems, but also to check various properties by representing requirements with a suitable query. We show the feasibility of our approach with experimentation on standard benchmarks taken from the hybrid automata context.

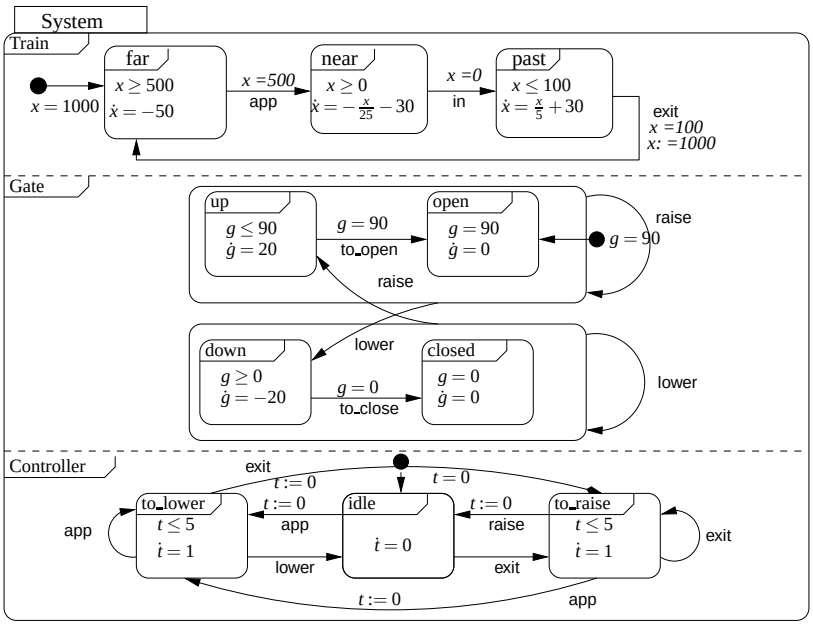

Fig. 1. Specification of the train gate controller as hybrid automata.

1.1 Overview on the Rest of the Paper

In summary, the main contributions of this paper are as follows: First, an effective framework, implemented in CLP, is presented, which is suitable to model and verify multi-gent systems based on hybrid automata. Second, compositions of automata do not have to be computed explicitly prior to verifying multi-agent systems. Instead, the composition of automata is built dynamically during the verification phase, which can relieve the state explosion problem that may raise as the result of multi-agent systems. Last but not least, by employing CLP, constraints can be derived automatically, under which certain states of a system can be tested for reachability. This enhances standard model checking methodologies.

In the sequel, we first introduce a running example that will be used throughout the paper to illustrate our approach in Sec. 2. Then hybrid automata syntax and semantics are discussed in Sec. 3. In Sec. 4 a CLP implementation model is discussed, before showing how to specify and verify requirements in Sec. 5. The evaluation of our CLP implementation model is discussed in Sec. 6. Then Sec. 6.1 briefly reviews related works, before we end up with the conclusion Sec. 7

2 Running Example

Before we present both syntax and semantic of hybrid automata, we first introduce an illustrating running example that we use throughout the paper, before we shows the basics formalism which we use to demonstrate the CLP implementation.

A train gate controller [14] is a reactive multi-agent system consisting of three agent components: the train, the gate, and the controller. In this system, a road is crossing a train track, which is guarded by a gate, which must be lowered to stop the traffic when the train approaches, and raised after a train passed the road. The gate is supervised by a controller that has the task to receive signals from the train and to issue lower or raise signals to the gate. Initially, a train is at a distance of 1000 meters away from the gate and moves at a speed 50 meter per second. At 500 meters, a sensor on the tracks detects the train, sending a signal *app* to the controller. The train slows down, obeying the differential equation $\dot{x} = -\frac{x}{25} - 30$. After a delay of five seconds, which is modeled by the variable *t*, the controller sends the signal *lower* to the gate, which begins to descend from 90 degrees to 0 degrees at a rate of -20 degrees per second. After crossing the gate, the train accelerates according to the differential equation $\dot{x} = \frac{x}{5} + 30$. A second sensor placed 100 meters past the crossing detects the leaving train, sending a signal *exit* to the controller. After five seconds, the controller raises the gate.

The specification of the previous multi-agent system is graphically illustrated as concurrent hybrid automata in Fig. 1. The variable *x* represents the distance of the train from the gate. The variable *t* represents the delay time of the controller, while the position of the gate in radius degrees is represented by the variable *g*.

3 Hybrid Automata Preliminaries

In this section, we show the basics syntax and the semantics of hybrid automata.

3.1 Hybrid Automaton: Syntax

A hybrid automaton is represented graphically as a state transition diagram dialect like statecharts, augmented with mathematical formalisms on both transitions and locations. Formally speaking, a hybrid automaton (agent in continuous domain) is defined as follows.

Definition 1 (basic components). *A hybrid automaton is a tuple* $H = (X, Q, Inv, Flow, E, Jump, Reset, Event, Init)$ *where:*

- $X \subseteq \Re^n$ *is a finite set of n real-valued variables that model the continuous dynamics.*
- *Q is a finite set of control locations. For example, the train automaton (Fig. 1) has the locations far, near,and past.*
- $Inv(q)$ *is the invariant predicate, which assigns a constraint on variables X for each control location* $q \in Q$*. The control of a hybrid automaton remains at a location* $q \in Q$*, as long as* $Inv(q)$ *holds. For instance, the location* far *in the train automaton has the invariant* $x \geq 500$
- $Flow(q)$ *is the flow predicate on variables X for each control location* $q \in Q$*, which defines how the the variables in X evolve over the time at location q. It constrains the time derivative of the continuous part of the variables at location q (Basically, we represent the flow as a constrain relation of the real variables to the time). In the graphical representation, a flow of a variable x is denoted as* $\dot{x}$*. For example,* $\dot{x} = \frac{x}{5} + 30$ *describes the speed of the train at the location* past *in the train automaton*

(Fig. 1). If $\dot{x} = c$, then the hybrid automaton is called linear (a special case of linear hybrid automata are a timed automata [2], where $c = 1$). if $\dot{x} = c_1 x + c_2$, then a hybrid automaton is called non-linear.

- *$E \subseteq Q \times Q$ is the discrete transition relation over the control locations. Each edge $e \in E$ is augmented by the following annotations:*
  - **Jump:** *jump condition (guard), which is a constraint over X that must hold to fire transitions. Omitting a jump condition on a transition means that the jump condition is always true and it can be taken at any point of time. Conventionally, writing $Jump(e)[v]$ means that the jump condition on a transition e holds, if the valuations of variables on the transition are v.*
  - **Reset:** *is a constraint, which may reset the variables by executing a specific assignments. For example, the variable X in the train automaton on the transition between locations* past *and* far *is reset to $X := 1000$. Resetting variables are omitted on transition, if the values of the variables do not change before the control goes from a location to another location.*
  - **Event:** *synchronization label, used to synchronize concurrent automata. For instance, the train automaton contains the synchronization labels* app, in, *and* exist, *which must be synchronized with all automata sharing the same synchronization labels. These synchronization labels define the composition of the automata.*
- *Init is the initial condition that assigns an initial values to the variables X to each control location $q \in Q$. For example, $x = 1000$ is the initial condition of the train automaton.*

3.2 Hybrid Automaton: Semantics

Informally speaking, the semantics of a hybrid automaton is defined in terms of a labeled transition system between states, where a state consists of the current location of the automaton and the current valuation of the real variables. To formalize the semantics of the hybrid automaton, first we need to define the concept of a hybrid automaton's state.

Definition 2 (State). *At any instant of time, a state of a hybrid automaton is given by $\sigma_i = \langle q_i, v_i, t \rangle$, where $q_i \in Q$ is a control location, v_i is the valuation of its real variables, and t is the current time. A state $\sigma_i = \langle q_i, v_i, t \rangle$ is admissible if $Inv(q_i)[v_i]$ holds.*

A state transition system of a hybrid automaton H starts with the *initial state* $\sigma_0 = \langle q_0, v_0, 0 \rangle$, where the q_0 and v_0 are the initial location and valuations of the variables respectively. For example, the initial state of the *train* (see Fig. 1) can be specified as $\langle far, 1000, 0 \rangle$.

In fact, a hybrid automaton evolves depending on two kinds of transitions: continuous transitions, capturing the continuous evolution of states, and discrete transitions, capturing the changes of location. More formally, we can define hybrid automaton semantics as follows.

Definition 3 (Operational Semantic). *A transition rule between two admissible states $\sigma_1 = \langle q_1, v_1, t_1 \rangle$ and $\sigma_2 = \langle q_2, v_2, t_2 \rangle$ is defined as follows:*

discretely: *iff $e = (q_1, q_2) \in E$, $t_1 = t_2$ and $Jump(e)[v_1]$ holds, then variables are reset according to v_2 such that $Inv(q_2)[v_2]$ holds at location q_2. In this case an event $a \in Event$ occurs. Conventionally, it is written $q_1 \xrightarrow{a} q_2$.*

continuously(time delay): *iff $q_1 = q_2$, and $(t_2 - t_1 > 0)$ is the duration of time passed at location q_1, during which the invariant predicate $Inv(q1)$ continuously holds, v_1 and v_2 are the valuations of the variables according to the flow predicate $Flow(q1)$.*

Intuitively, an execution of a hybrid automaton corresponds to a sequence of transitions from a state to another. Therefore we define the valid run as follows.

Definition 4 (Run:Micro level). *A run of hybrid automaton $\sum = \sigma_0\sigma_1\sigma_2,..,$ is a finite or infinite sequence of admissible states, where σ_0 is the initial state.*

In the run $\sum$, the transition from a state σ_i to a state σ_{i+1} is related by either a discrete or a continuous transition according to Def. 3. It should be noted that the continuous change in the run may generate an infinite number of reachable states. It follows that state-space exploration techniques require a symbolic representation way in order to represent the set of states in a appropriate way. In this paper, we use CLP to represent the infinite states symbolically as finite intervals. we call a symbolic interval as a region, which is defined as follows:

Definition 5 (Region). *A region $\Gamma = \langle q, V, Time \rangle$ is the set of possible states reached at location q by means of continuous transitions, where V and $Time$ represent interval of reached valuations of the variables together with their reached time at location q respectively. A region Γ is admissible if $inv(q)[v]$ holds for all $v \in V$.*

Now, the run of hybrid automata can be rephrased in terms of reached regions, where the change from one region to another is fired using a discrete step.

Definition 6 (Run:Macro level). *A run of hybrid automaton H is $\sum_H = \Gamma_0\Gamma_1,...$ a sequence of (possibly infinite) admissible regions, where a transition from a region Γ_i to a region Γ_{i+1} is enabled (written as $\Gamma_i \xrightarrow{a} \Gamma_{i+1}$), if there is $q_i \xrightarrow{a} q_{i+1}$, where $a \in Event$ is the generated event before the control goes to the region Γ_{i+1}. Γ_0 is the initial region reached from a start state σ_0 by means of continuous transitions.*

The operational semantics is the basis for verification of a hybrid automaton. In particular, model checking of a hybrid automaton is defined in terms of the reachability analysis of its underlying transition system. The most useful question to ask about hybrid automata is the reachability of a given state. Thus, we define the reachability of states

Definition 7 (Reachability). *A region Γ_i is called reachable in $\sum_H$, if $\Gamma_i \subseteq \sum_H$. Consequently, a state σ_j is called reachable, if there is a reached region Γ_i such that $\sigma_j \in \Gamma_i$*

The classical method to compute the reachable states consists of performing a state space exploration of the system, starting from a set containing only the initial state and spreading the reachability information along control locations and transitions until a stable region is obtained. Stabilization is detected by testing if the current region is included in the union of the reached regions obtained in previous steps. It is worth mentioning that checking reachability for hybrid automata is generally undecidable. It is decidable, However, for certain classes of hybrid automaton [15].

3.3 Hybrid Automata: Composition

To specify complex systems, hybrid automata can be extended by parallel composition. Basically, the parallel composition of hybrid automata can be used for specifying larger systems (multi-agent systems), where a hybrid automaton is given for each part of the system, and communication between the different parts may occur via shared variables and synchronization labels. Technically, the parallel composition of hybrid automata is obtained from the different parts using a product construction of the participating automata. The transitions from the different automata are interleaved, unless they share the same synchronization label. In this case, they are synchronized during the execution. As a result of the parallel composition, a new automaton, called composed automaton, is created, which captures the behavior of the entire system. In turn, the composed automata are given to a model checker that checks the reachability of a certain state.

Intuitively, the composition of hybrid automata H_1 and H_2 can be defined in terms of synchronized or interleaved regions of from the regions produced from run of both H_1 and H_2. As a result from the composition procedure, compound regions are constructed that consists of a conjunction of one region from H_1 and another from H_2. Therefore, each compound region takes the form $\Lambda = \langle (q_1, V_1), (q_2, V_2), T \rangle$ (shortly written as $\Lambda = \langle \Gamma_1, \Gamma_2, T \rangle$), which represents reached region at both control locations q_1 and q_2 the during a time interval T. Now the run of composed automata is the sequence $\sum_{H_1 \circ H_2} = \Lambda_0, \Lambda_1, ...$, where a transition between compound region $\Lambda_1 = \langle \Gamma_1, \gamma_1, T_1 \rangle$ and $\Lambda_2 = \langle \Gamma_2, \gamma_2, T_2 \rangle$ (written as $\Lambda_1 \xrightarrow{a} \Lambda_2$) is enabled, if one of the following holds:

- $a \in Event_{H_1} \cap Event_{H_2}$ is a joint event, $\Gamma_1 \xrightarrow{a} \Gamma_2$, and $\gamma_1 \xrightarrow{a} \gamma_2$. In this case , we say that the region Γ_1 is synchronized with the region γ_1.
- $a \in Event_{H_1} \setminus Event_{H_2}$ (respectively $a \in Event_{H_2} \setminus Event_{H_1}$), $\Gamma_1 \xrightarrow{a} \Gamma_2$ and $\gamma_1 \rightarrow \gamma_2$, such that both γ_1 and γ_2 have the same control location (i.e., they relate to each other using a continuous transition).

The previous procedures give the possibility to construct the composition dynamically during the run/verification phase. Obviously, computing the composition in such a way is advantageous. This is for the reason that the only the active parts of the state space will be taken into consideration during the run, instead of producing the composition procedure prior to verification phase. This can relieve the state space problem raised from modeling multi-agent systems.The coming section shows how the previous procedure, with the help of constraint logic programming, can be performed.

4 CLP Model

In the following, we will show how to encode the syntax and semantics of hybrid automata, described in the previous section, as a Constraint Logic Program *CLP* [19]. There are diverse motivations for choosing CLP. Firstly, hybrid automata can be described as a constraint system, where the constraints represent the possible flows, invariants, and transitions. Further, constraints can be used to characterize certain parts of the state space (e.g., the set of initial state or a set of unsafe state). Secondly, there

are close similarities in operation semantics between CLP and hybrid automata. Ideally, state transition systems can be represented as a logic program, where the set of reachable states can be computed. Moreover, constraints enable us to represent infinite states symbolically as a finite interval. Hence, the constraint solver can be used to reason about the reachability of a particular state. In addition, CLP is enriched with many efficient constraint solvers for interval constraints and symbolic domains, where the interval constraints can used to represent the continuous evolution, whereas symbolic domains are appropriate to represent the synchronization events (communication messages).

Our implementation prototype was built using ECLiPSe Prolog [21]. A preliminary implementation model was introduced in [25]. The prototype follows the definitions of both the formal syntax and semantics of hybrid automata, which are defined in the previous section. We start modeling each hybrid automaton individually. Therefore, we begin with modeling locations of automata that are implemented in the `automaton` predicate, ranging over the respective locations of the automaton, real-valued variables and the time:

```
automaton(+Location,?Vars,+Vars0,+T0,?Time):-
        Vars#c2(Vars0,(Time-T0)),
        c1(Inv),Time $>=T0.
```

Here, `automaton` is the name of automaton itself, and `Location` represents the ground name of the current locations of the automaton. `Vars` is a list of real variables participating in the automata, whereas `Vars0` is a list of the correspondent initial values. `c1(Invs)` is the invariant constraint on `Vars` inside the location. The constraint predicate `Vars # c2(Vars0,(Time-T0))`, where #$\in \{<, \leq, >, \geq, =\}$ are constraints, which represent the continuous flows of the variables in `Vars` wrt. time `T0` and `Time`, given initial values `Vars0` of the variables `Vars` at the start of the flow. `T0` is the initial time at the start of the continuous flow, while `(Time-T0)` represents the delay inside the location. The following is an example showing the concrete implementation of location *far* in the automaton *train* Fig. 1. The $ symbol in the front of (in)equalities is the constraint relation for interval arithmetic constraints (library *ic* in ECLiPSe Prolog).

```
train(far,[X],[X0],T0,Time):-
     X $= X0-50*(Time-T0),
     X $>=500, Time $>=T0.
```

According to operational semantics defined in Def. 3, a hybrid automaton has two kinds of transitions: *continuous* transitions, capturing the continuous evolution of variables, and *discrete* transitions, capturing the changes of location. For this purpose, we encode transition systems into the predicate *evolve*, which alternates the automaton between a discrete and a continuous transition. The automaton evolves with either discrete or continuous according to the constraints appeared during the run.

```
evolve(+Automaton,(+L1,+Var1),(+L2,+Var2),+T0,+Time,-Event) :-
     continuous(Automaton,(L1,Var1),(L1,Var2),T0,Time,Event);
     discrete(Automaton,(L1,Var1),(L2,Var2),T0,Time,Event).
```

When a discrete transition occurs, it gives rise to update the initial variables from `Var1` into `Var2`, where `Var1` and `Var2` are the initial variables of locations `L1` and `L2` respectively. Otherwise, a delay transition is taken using the predicate *continuous*. It is worth noting that there are infinite states due to the continuous progress. However, this can be handled efficiently as interval constraint that bounds the set of infinite reachable state as a finite interval (i.e., $0 \leq X \leq 250$).

In addition to the variables, each automaton is augmented with set of events, which we call it $\in Event_{Automaton}$. For example, $Event_{train} = \{app, in, exit\}$. Fro this reason, each transition is augmented with the variable `Event`, which is used to define the parallel composition from the automata individuals sharing the same event. The variable `Event` ranges over symbolic domains. It guarantees that whenever an automaton generates an event, the corresponding synchronized automata have to be taken into consideration simultaneously. It should be mentioned that the declaration of automata events must be provided in the modeling example. For instance, the declaration of the possible events domains of Fig. 1. are coded as follows :

```
:- local domain(events(app,in,exit,raise,lower, to_open)).
```

The previous means that the domains of events are declared symbolically to capture the set of all possible applicable events to the underlying modeled system. The appropriate solver of symbolic domain deals with any constraints defined in terms of the declared domains. Now after defining the domains of events, a variable of type events can be declared as follow:

```
Event &::events,Event &=domain_value.
```

The previous means that a variable `Event` is declared with domain values defined by `events`, and is initialized with a specific value from its domain. The & symbol is a constraint relation for symbolic domains (library *sd* in ECLiPSe Prolog).

An automaton generates an event thanks to a discrete transition (generating an events means, the variable `Event` takes a value from its domain). This event has to be synchronized with the other automata sharing the same event. For this reason, each transition is augmented with the variable `Event`. This variable takes a value from its domain, during firing a discrete transition. The following is the general implementation of the predicate `discrete`, which defines transitions between locations.

```
discrete(+Automaton,(+Loc1,+Var1),(?Loc2,?Var2),+T0,+Time,-Event):-
         automaton,(Loc1,Var1,Var,T0,Time),
         jump(Var), reset(Var2),
         Event &::events,Event &=domain_value.
```

In the previous predicate, `domain_value` must be a member in $Event_{Automaton}$.

The following is an instance showing the implementation of the `discrete` predicate between locations *far* and *near* in automaton *train*.

```
discrete(train,(far,[X0]),(near,[XX0]),T0,Time,Event):-
         train(far,[X0],[X],T0,Time),
         X $=500, XX0 $=X,
         Event &::events, Event &=app.
```

The description of the previous *discrete* predicate means that the transition between the locations *far* and *near* in the *train* automata takes place, if the continuous variable X, based on the initial value X0, satisfies the jump condition given as X=500. If such a case occurs, then the new variable, denoted as XX0, is updated and the event *app* is fired. The executed events afterwards synchronize the *train* automaton with the automata sharing the same event.

Once the transition rules have been modeled, a driver program needs to be supplied:

```
driver((+L1,+Var01),(+L2,+Var02),...,(+Ln,+Var0n),+T0,
[(L1,L2,..,Ln,-Var1,-Var2,..,-Varn,-Time,-Event)|-NextRegion],+PastReg) :-
     automaton1(L1,Var1,Var01,T0,Time1),
     automaton2(L2,Var2,Var02,T0,Time2),
     ... ,
     automatonn(Ln,Varn,Var0n,T0,Timen),
     Time1 $=Time2, Time1 $=Time3, ..., Time1 $=Timen,
     evolve(automaton1,(L1,Var01),(NextL1,Nvar01),T0,Time1,Event),
     evolve(automaton2,(L2,Var02),(NextL2,Nvar02),T0,Time1,Event),
     ... ,
     evolve(automatonn,(Ln,Var0n),(NextLn,Nvar0n),T0,Time1,Event),

    +\member((L1,L2,..,Ln,Var1,Var2,..,Varn,_,Event), PastReg),
    NpastReg =[L1,L2,..,Ln,Var1,Var2,..,Varn,Time,Event)|PastReg],

driver((NextL1,Nvar01),(NextL2,Nvar02),...,(NextLn,Nvar0n),Time1,
                                          NextRegion,NpastReg).
```

The driver is a simulator predicate that is responsible to generate and control the behaviors of the concurrent hybrid automata, as well as to provide the reachable regions symbolically.

Inside the definition of the predicate driver, the variable Event is a symbolic domain variable shared among all automata. It is used by the appropriate solver to ensure that only one event is generated at a time, such that when an automaton generates an event, thanks to a discrete transition of one of the predicates evolve of the concurrent automata, then the symbolic domain solver will exclude all the domain values of the other automata that are not coincident with the generated event. This means that only one event is generated at a time. If it happens the case that more than one automaton generate different events at the same point of time, then the symbolic domain solver will handle only one of them at a time, but the other events will be handled using backtracking.

Since each automaton, at the end of its continuous evolution, generates an event, then the precedence of events that appear during the run are important to the composition and to the verification too. For this reason, an obvious way to deal with this precedence is to use constraints on the time of the generated events. To be precise, each automaton A_i ,$1 \leq i \leq n$, produces a time $Time_i$, which is needed to jump from the automaton's current location into another location. Constraining these times of each automaton together leads to a time holding the minimum time among them. This minimum time in this case, manipulated by the constraints solver, is least time needed to fire an event. The previous computation partitions the state space into regions, where the

transition from one region to another depends on the minimum time needed to generate an event. Consequently, this shows how the automata composition can be implicitly constructed efficiently on the fly, during the computation. Appropriately, the way that we construct the composition helps us to construct complex automata in terms of simpler ones.

The last argument of the predicate `driver` is the list of reached regions. At each step of the driver, a region, of the form $\langle locations, Variables, Time \rangle$ represents symbolically the set of reached states and times to each control location as mathematical constrains. Additionally, each region contains the event generated before the control goes to another region using a discrete step. Technically, the `driver` computes the set of reached regions until fixed regions are obtained. This is computed by checking, in each iteration of driver, if the reached region is not contained in the list of the previously reached regions. For this purpose, the `driver` should be augmented with an extra argument containing the list of past reached regions. Broadly speaking, the termination of the driver to reach to a fixed regions is not guaranteed. Fortunately, it does terminate for all the examples in the experimental result. However, to overcome the non termination problem generally, one can augment the predicate `driver` with some iteration depth in advance, where the driver is enforced to stop upon reaching this depth. .

Reachable regions should contain only those variables, which are important for the verification of a given property. Therefore, the last argument list of the predicate `driver` can be expanded or shrunk as needed to contain the significant variables.

The driver has to be invoked with a query starting from the initial states of the hybrid automata. An example showing how to query the driver on the running scenario (Fig. 1) takes the form:

```
?- driver((far,1000),(open,90),(idle,0),0,Reached,[]).
```

5 Verification as Reachability Analysis

Now we have an executable constraint based specification, which can be used to verify properties of a multi-agent system. Several properties can now be investigated. In particular, one can check properties on states using reachability analysis. Fundamentally, the reachability analysis consists of two basic steps. First, computing the state space of the automaton under consideration. In our case, this is done using the predicate `driver`. Second, searching for states that satisfy or contradict given properties. This is done using a standard prolog predicates like *member*/2 and *append*/3. Therefore, we present CLP rules, which constitute our verification framework. The validation of these rules depends totally on the set of reached regions that we described them formally in Sec.3, and implemented in Sec.4.

In terms of *CLP*, a state is reached iff the constraint solver succeeds in finding a satisfiable solution for the constraints representing the intended state. In other words, assuming that *Reached* represents the set of all reachable states computed by the *CLP* model from an initial state, then the reachability analysis can be generally specified, using *CLP*, by checking whether `Reached` $\models \Psi$ holds, where Ψ is the constraint predicate that describes a property of interest. In practice, many problems to be analyzed can be

formulated as a reachability problem. For example, a safety requirement can be checked as a reachability problem, where Ψ is the constraint predicate that describes forbidden states, and then the satisfiability of Ψ wrt. `Reached` is checked. For instance, one can check that the state, where the train is near at distance `X=0` and the gate is open, is a disallowed state. Even a stronger condition can be investigated, namely that the state where the train is near at distance `X=0` and the gate is down, is a forbidden state. The *CLP* computational model, with the help of the standard Prolog predicate *member*/2, gives us the answer *no* as expected, after executing the following query:

```
?- driver((far,1000),(open,0),(idle,0),0,Reached,[]),
   member((near,down,_,Time,_,X,),Reached), X $= 0.
```

Other properties concerning the reachability of certain states can be verified similarly.

Fundamentally, different properties can be checked in this framework As previously demonstrated, the set of reachable states `Reached` contains the set of finite, reachable regions. Within each region, the set of all states is represented symbolically as a mathematical constraint, together with the time delay. Therefore, constraint solvers ideally can be used to reason about the reachability of interesting properties within some region. For example, an interesting property is to find the shortest distance of the train to the gate before the gate is entirely closed. This can be checked by posing the following query:

```
?- driver((far,1000),(open,0),(idle,0),0,Reached.[]),
   member((near,_,_,Time,to_close,_),Reached), get_max(Time,Tm),
   member((near,_,_,Tm,_,X),Reached), get_min(X,Min).
```

The previous query returns `Min=104.8` meters, which is the minimum distance of the train that the model guarantees before the gate is completely closed.

Since the events and time are recorded particularly at reached regions, verifying timing properties or computing the delay between events are further tasks that can be done within the reachability framework too. For instance, we can find the maximal time delay between *in* and *exit* events, by stating the following query:

```
?- driver((far,1000),(open,0),(idle,0),Reached,[]),
      append(A,[(past,_,_,Time1,exit,_)|_],Reached),
      append(B,[(near,_,_,Time2,in,_)|_],A),
      get_max(Time1,Tmax1),get_max(Time2,Tmax2),
      Delay $= Tmax1-Tmax2.
```

The constraint solver answers `yes` and yields `Delay=2.554`. This value means that the train needs at most `2.554` seconds to be in the critical crossing section before leaving it. Similarly, other timing properties can be verified.

6 Experimental Results

In the previous section, we have demonstrated how different properties can be verified within the *CLP* implementation framework.

We did several experiments comparing our approach with HyTech [16]. We chose HyTech as a reference tool, because it is one of the most well-known tools for the verification of hybrid automata, and it tackles verification based on reachability analysis

similar to the approach in this paper. In HyTech however, the automata working in parallel are composed before they are involved in the verification phase. Obviously, this may lead to state explosion as stated earlier.

Now to use our approach to model and verify multi-agent systems, specified as hybrid automata, we have to demonstrate the feasibility of our proposed approach by experiments taken from the hybrid automata context. Therefore, we will refer to standard benchmarks of verification of real-time systems. Querying these benchmarks to check safety properties (cf. Fig. 2). First, in the *scheduler* example [11], it is checked whether a certain task (with number 2) never waits. Second, in the *temperature control* example [1], it has to be guaranteed, that the temperature always lies in a given range. Third, in the *train gate controller1* example [13], it has to be ensured that the gate is closed whenever the train is within a distance less than 10 meter toward the gate. In the *water level* example [1, 11] the safety property is to ensure that the water level is always between given thresholds (1 and 12). A non-linear version of both train gate controller (described throughout this paper) and of the thermostat are taken from [14]. The safety property of the former one is the same as in the linear version, whereas in the second one we need to prove that the temperature always lies between 0.28 and 3.76. Last but not least, nuclear *Reactor* examples are taken from the verification examples of HyTech [16]. The safety property of both example is to ensure that only one of the rods of the reactor can be put in. For more details on the examples, the reader is referred to the cited literature for more details.

Example	**HyTech**	**CLP**
Scheduler	0.12	0.07
Temperature Controller	0.04	0.02
Train Gate Controller1	0.05	0.02
Water Level	0.03	0.01
Train Gate Controller2	-	0.02
Thermostat	-	0.01
Reactor1	0.01	0.01
Reactor2	-	0.01

Fig. 2. Experimental results.

The symbol $-$ in Fig. 2 indicates that the example is inadequate to HyTech. This is because HyTech can not treat a non-linear dynamic directly. Instead, It checks approximation versions of these examples.

When comparing HyTech to the approach depicted in this paper, several issues have to be taken into consideration. The first issue concerns the expressiveness of the dynamical model. HyTech restricts the dynamical model to linear hybrid automata in which the continuous dynamics is governed by differential equations. The nonlinear dynamics e.g. of the form $\dot{x} \bowtie c1 * x + c2$, where $c1, c2 \in \Re$,$c1 \neq 0, \bowtie \in \{<, \leq, >, \geq, =\}$ are first approximated either by a linear phase portrait or clock translation [17]. Then, the verification phase is done on the approximated model. On the other hand, *CLP* is more expres-

sive, because it allows more general dynamics. In particular, *CLP* can directly handle dynamics expressible as a combination of polynomials, exponentials, and logarithmic functions explicitly without approximating the model. For instance the last equation can be represented in *CLP* form as $X\ \$\bowtie X0 - c2/c1 + c2/c1 * exp(c1 * (T - T0))$, where $(T - T0)$ is the computational delay. Although clearly completeness cannot be guaranteed, from a practical point of view, this procedure allows to express problems in a natural manner. The *CLP* technology can be fully exploited; it suspends such complex goals until they become solvable.

Another issue that should be taken into account is the type of verifiable properties. HyTech cannot verify simple properties that depend on the occurrence of events, despite of the fact that synchronization events are used in the model. On the other hand, simple real-time duration properties between events can be verified using HyTech. However, to do so, the model must be specified by introducing auxiliary variables to measure delays between events or the delay needed for a particular conditions to be hold. Bounded response time and minimal event separation are further properties that can be verified using HyTech. These properties, however, can only be checked after augmenting the model under consideration with what is called a *monitor* or *observer* automaton (cf. [13]), whose functionality is to observe the model without changing its behavior under consideration. It records the time as soon as some event occurs. Before the model is verified, the monitor automaton has to be composed with the original model, which in turns may add further complexity to the model. As demonstrated in this paper, however, there is no need to augment the model with an extra automaton for the reason that during the run, not only the states of variables are recorded, but also the events and the time, where the constraint solver can be used to reason about the respective property

In addition to the benchmarks that demonstrate the feasibility of our approach, we have recently used our framework to model a case study that has been taken from a logistic domain [23]. In this case study, we have demonstrated how an agent, in a multi-agent scenario, selects the most appropriate plan, in case of occurring unexpected events during the plan's execution. The agent selects the plan that maximizes its utility function. The expressiveness of classical tools of hybrid automata lack to model such types of scenario. This is for the reason that these tools are special purpose tools that model continuous reactive systems. The Expressive of hybrid automata on multi-agent system, in terms of modeling and verification, are not the main concerns of these tools.

6.1 Related Works

Since we have presented and implemented an approach to model and verify multi-agent systems by means of hybrid automata, this section will relate our work to the other approaches of hybrid automata. The key relation to these approaches to multi-agent systems is that all of them can be used to model and validate multi-agent systems plans that are defined through their capability to continuously react in dynamic environments, while respecting some time constraints.

Several tools exist for formal verification of hybrid automata [13, 8, 3], where a multi-agent team can be verified. Differently to our approach, however, these tools compose the automata prior to the verification phase.

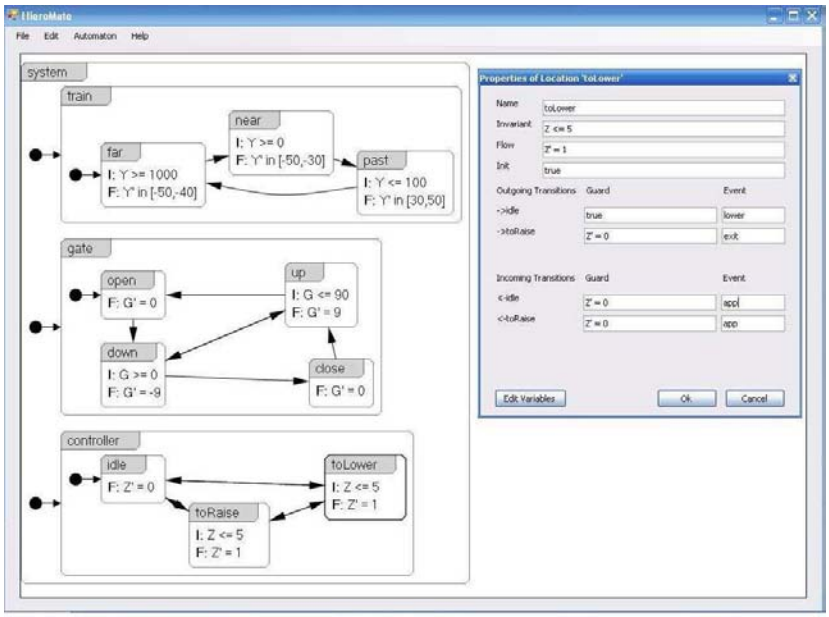

Fig. 3. A tool for modeling and Verification based on CLP.

We are not the first one who approached modeling and verifying hybrid automata using CLP. In contrast to our proposed approach, several authors propose the explicit composition of different concurrent automata by hand leading to one single automaton, before a CLP implementation is applied. This is a tedious work, especially in the case of multi-agent systems, where a group of agents exists. The latter case is exemplified in [27, 20]. Other authors employ CLP for implementing hybrid automata [4, 5, 10], but restrict their attention to a simple class of hybrid systems (e.g. timed systems). They do not construct the overall behavior prior to modeling, but model each automaton separately. However, the run of the model takes all possible paths into consideration, resulting from the product of each component, which leads to unnecessary computation.

7 Conclusion

Multi-agent systems need to coordinate their plans especially in a safety critical environment, where unexpected events typically arise. Therefore, it is becoming increasingly important to react to those events in real time in order to avoid the risk that may occur during the planning. For this purpose, various researches have approached hybrid automata as a framework to model reactively multi-agent plans. In this paper, we have showed how multi-agent systems can be formally specified and verified as hybrid automata without explicitly composing the system prior to the verification phase. The previous helps to tackle the state space problem that may arise during the composition

process. We have programmed our approach by means of constraint logic programming, where constraint solvers help us to build dynamically the entire behavior of a multi-agent system and to reason about its properties. Furthermore, we have showed how various properties can be verified using our CLP framework. In addition, we have conducted several experiments taken from the hybrid automata context to show the feasibility of our approach.

Currently we are developing and enhancing a tool environment that aims at simplifying both processes of modeling and verification. In this tool, a model together with its requirement are specified graphically, then the process of verification is achieved automatically. The graphical specifications are transformed into executable CLP codes, which follows the outline of this paper. This can avoid the tedious work, which results from specifying larger systems. Additionally, this also give the possibility to the non experts people of CLP to model and verify multi-agent systems based on hybrid automata. A primary version of the tool (see Fig.3) appears in [24]. In addition to the tool, since CLP is a suitable framework, where we can reason not only about the time behaviors of multi-agent systems, bout also about their knowledge, then the combination of both worlds is subjected to a future work.

References

1. R. Alur, C. Courcoubetis, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. In *ICAOS: International Conference on Analysis and Optimization of Systems – Discrete-Event Systems*, Lecture Notes in Control and Information Sciences 1994, pages 331–351. Springer, Berlin, Heidelberg, New York, 1994.
2. R. Alur and D. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
3. G. Behrmann, A. David, and K. G. Larsen. A tutorial on Uppaal. In M. Bernardo and F. Corradini, editors, *Proceedings of 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems – Formal Methods for the Design of Real-Time Systems (SFM-RT)*, LNCS 3185, pages 200–236. Springer, Berlin, Heidelberg, New York, 2004.
4. A. Ciarlini and T. Frühwirth. Automatic derivation of meaningful experiments for hybrid systems. *Proceeding of ACM SIGSIM Conf. on Artificial Intelligence, Simulation, and Planning (AIS'00)*, 2000.
5. G. Delzanno and A. Podelski. Model checking in CLP. *Proceeding, Tools and Algorithms for Construction and Analysis of Systems (TACAS'99)*, pages 223–239, 1999.
6. M. Egerstedt. Behavior Based Robotics Using Hybrid Automata. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 103–116, 2000.
7. A. El Fallah-Seghrouchni, I. Degirmenciyan-Cartault, and F. Marc. Framework for Multi-agent Planning Based on Hybrid Automata. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 226–235, 2003.
8. G. Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. In M. Morari and L. Thiele, editors, *Hybrid Systems: Computation and Control, 8th International Workshop, Proceedings*, LNCS 3414, pages 258–273. Springer, 2005.
9. U. Furbach, J. Murray, F. Schmidsberger, and F. Stolzenburg. Hybrid multiagent systems with timed synchronization – specification and model checking. In M. Dastani, A. El Fallah Seghrouchni, A. Ricci, and M. Winikoff, editors, *Post-Proceedings of 5th International*

Workshop on Programming Multi-Agent Systems at 6th International Joint Conference on Autonomous Agents & Multi-Agent Systems, LNAI 4908, pages 205–220. Springer, 2008.
10. G. Gupta and E. Pontelli. A constraint-based approach for specification and verification of real-time systems. *Proceedings of IEEE Real-time Symposium*, pages 230–239, 1997.
11. N. Halbwachs, Y. Proy, and P. Raymond. Verification of linear hybrid systems by means of convex approximations. In *Static Analysis – Proceedings of 1st International Static Analysis Symposium (SAS'94)*, LNCS 864, pages 223–223, Namur, Belgium, 1994. Springer, Berlin, Heidelberg, New York.
12. T. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, pages 278–292, New Brunswick, NJ, 1996. IEEE Computer Society Press.
13. T. Henzinger, P.-H. Ho, and H. Wong-Toi. A user guide to HyTech. In *Proceedings of International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS 1019, pages 41–71. Springer, Berlin, Heidelberg, New York, 1995.
14. T. Henzinger, B. Horowitz, R. Majumdar, and H. Wong-Toi. Beyond HYTECH: Hybrid Systems Analysis Using Interval Numerical Methods. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 130–144, 2000.
15. T. Henzinger, P. Kopke, A. Puri, and P. Varaiya. What's Decidable about Hybrid Automata? *Journal of Computer and System Sciences*, 57(1):94–124, 1998.
16. T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: The Next Generation. In *IEEE Real-Time Systems Symposium*, pages 56–65, 1995.
17. T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43:540–554, 1998.
18. G. Hutzler, H. Klaudel, and D. Y. Wang. Towards timed automata and multi-agent systems. In *Formal Approaches to Agent-Based Systems, Third InternationalWorkshop, FAABS 2004, Greenbelt, MD, USA, April 26-27, 2004, Revised Selected Papers*, volume 3228 of *Lecture Notes in Computer Science*, pages 161–172. Springer, 2005.
19. J. Jaffar and J. Lassez. Constraint logic programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 111–119. ACM New York, NY, USA, 1987.
20. J. Jaffar, A. Santosa, and R. Voicu. A clp proof method for timed automata. *Real-Time Systems Symposium, IEEE International*, 0:175–186, 2004.
21. M. W. Krzysztof R. Apt. *Constraint Logic Programming Using Eclipse*. Cambridge University Press, Cambridge, UK, 2007.
22. A. Mohammed and U. Furbach. Modeling multi-agent logistic process system using hybrid automata. In U. Ultes-Nitsche, D. Moldt, and J. C. Augusto, editors, *MSVVEIS 2008: Proceedings of the 6th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems, MSVVEIS-2008*, pages 141–149. INSTICC PRESS, 2008.
23. A. Mohammed and U. Furbach. From reactive to deliberative multi-agent planning. In U. Ultes-Nitsche, D. Moldt, and J. C. Augusto, editors, *In Proceedings of the 7th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems, MSVVEIS 2009*, pages 67–75. INSTICC PRESS, 2009.
24. A. Mohammed and C. Schwarz. Hieromate: A graphical tool for specification and verification of hierarchical hybrid automata. In *KI 2009: Advances in Artificial Intelligence, Proceedings of the 32nd German Conference on Artificial Intelligence*, LNAI. to appear.
25. A. Mohammed and F. Stolzenburg. Implementing hierarchical hybrid automata using constraint logic programming. In S. Schwarz, editor, *Proceedings of 22nd Workshop on (Constraint) Logic Programming*, pages 60–71, Dresden, 2008. University Halle Wittenberg, Institute of Computer Science. Technical Report 2008/08.

26. Object Management Group, Inc. *UML Version 2.1.2 (Infrastructure and Superstructure)*, November 2007.
27. L. Urbina. Analysis of hybrid systems in CLP(R). In *Proceedings of 2nd International Conference on Principles and Practice of Constraint Programming (CP'96)*, LNAI 1118, pages 451–467, 1996.

Bisher erschienen

Arbeitsberichte aus dem Fachbereich Informatik

(http://www.uni-koblenz.de/fb4/publikationen/arbeitsberichte)

Multi-agent systems: Modeling and Virification using Hybrid Automata, Arbeitsberichte aus dem Fachbereich Informatik 10/2009

Performance Measurement auf der Basis von Kennzahlen aus betrieblichen Anwendungssystemen: Entwurf eines kennzahlengestützten Informationssystems für einen Logistikdienstleister, Arbeitsberichte aus dem Fachbereich Informatik 9/2009

Process Commodities: Entwicklung eines Reifegradmodells als Basis für Outsourcingentscheidungen, Arbeitsberichte aus dem Fachbereich Informatik 8/2009

Open-Source-Software für das Enterprise Resource Planning, Arbeitsberichte aus dem Fachbereich Informatik 7/2009

Ammar Mohammed, Frieder Stolzenburg, Using Constraint Logic Programming for Modeling and Verifying Hierarchical Hybrid Automata, Arbeitsberichte aus dem Fachbereich Informatik 6/2009

Tobias Kippert, Anastasia Meletiadou, Rüdiger Grimm, Entwurf eines Common Criteria-Schutzprofils für Router zur Abwehr von Online-Überwachung, Arbeitsberichte aus dem Fachbereich Informatik 5/2009

Hannes Schwarz, Jürgen Ebert, Andreas Winter, Graph-based Traceability – A Comprehensive Approach. Arbeitsberichte aus dem Fachbereich Informatik 4/2009

Anastasia Meletiadou, Simone Müller, Rüdiger Grimm, Anforderungsanalyse für Risk-Management-Informationssysteme (RMIS), Arbeitsberichte aus dem Fachbereich Informatik 3/2009

Ansgar Scherp, Thomas Franz, Carsten Saathoff, Steffen Staab, A Model of Events based on a Foundational Ontology, Arbeitsberichte aus dem Fachbereich Informatik 2/2009

Frank Bohdanovicz, Harald Dickel, Christoph Steigner, Avoidance of Routing Loops, Arbeitsberichte aus dem Fachbereich Informatik 1/2009

Stefan Ameling, Stephan Wirth, Dietrich Paulus, Methods for Polyp Detection in Colonoscopy Videos: A Review, Arbeitsberichte aus dem Fachbereich Informatik 14/2008

Tassilo Horn, Jürgen Ebert, Ein Referenzschema für die Sprachen der IEC 61131-3, Arbeitsberichte aus dem Fachbereich Informatik 13/2008

Thomas Franz, Ansgar Scherp, Steffen Staab, Does a Semantic Web Facilitate Your Daily Tasks?, Arbeitsberichte aus dem Fachbereich Informatik 12/2008

Norbert Frick, Künftige Anfordeungen an ERP-Systeme: Deutsche Anbieter im Fokus, Arbeitsberichte aus dem Fachbereicht Informatik 11/2008

Jürgen Ebert, Rüdiger Grimm, Alexander Hug, Lehramtsbezogene Bachelor- und Masterstudiengänge im Fach Informatik an der Universität Koblenz-Landau, Campus Koblenz, Arbeitsberichte aus dem Fachbereich Informatik 10/2008

Mario Schaarschmidt, Harald von Kortzfleisch, Social Networking Platforms as Creativity Fostering Systems: Research Model and Exploratory Study, Arbeitsberichte aus dem Fachbereich Informatik 9/2008

Bernhard Schueler, Sergej Sizov, Steffen Staab, Querying for Meta Knowledge, Arbeitsberichte aus dem Fachbereich Informatik 8/2008

Stefan Stein, Entwicklung einer Architektur für komplexe kontextbezogene Dienste im mobilen Umfeld, Arbeitsberichte aus dem Fachbereich Informatik 7/2008

Matthias Bohnen, Lina Brühl, Sebastian Bzdak, RoboCup 2008 Mixed Reality League Team Description, Arbeitsberichte aus dem Fachbereich Informatik 6/2008

Bernhard Beckert, Reiner Hähnle, Tests and Proofs: Papers Presented at the Second International Conference, TAP 2008, Prato, Italy, April 2008, Arbeitsberichte aus dem Fachbereich Informatik 5/2008

Klaas Dellschaft, Steffen Staab, Unterstützung und Dokumentation kollaborativer Entwurfs- und Entscheidungsprozesse, Arbeitsberichte aus dem Fachbereich Informatik 4/2008

Rüdiger Grimm: IT-Sicherheitsmodelle, Arbeitsberichte aus dem Fachbereich Informatik 3/2008

Rüdiger Grimm, Helge Hundacker, Anastasia Meletiadou: Anwendungsbeispiele für Kryptographie, Arbeitsberichte aus dem Fachbereich Informatik 2/2008

Markus Maron, Kevin Read, Michael Schulze: CAMPUS NEWS – Artificial Intelligence Methods Combined for an Intelligent Information Network, Arbeitsberichte aus dem Fachbereich Informatik 1/2008

Lutz Priese,Frank Schmitt, Patrick Sturm, Haojun Wang: BMBF-Verbundprojekt 3D-RETISEG Abschlussbericht des Labors Bilderkennen der Universität Koblenz-Landau, Arbeitsberichte aus dem Fachbereich Informatik 26/2007

Stephan Philippi, Alexander Pinl: Proceedings 14. Workshop 20.-21. September 2007 Algorithmen und Werkzeuge für Petrinetze, Arbeitsberichte aus dem Fachbereich Informatik 25/2007

Ulrich Furbach, Markus Maron, Kevin Read: CAMPUS NEWS – an Intelligent Bluetooth-based Mobile Information Network, Arbeitsberichte aus dem Fachbereich Informatik 24/2007

Ulrich Furbach, Markus Maron, Kevin Read: CAMPUS NEWS - an Information Network for Pervasive Universities, Arbeitsberichte aus dem Fachbereich Informatik 23/2007

Lutz Priese: Finite Automata on Unranked and Unordered DAGs Extented Version, Arbeitsberichte aus dem Fachbereich Informatik 22/2007

Mario Schaarschmidt, Harald F.O. von Kortzfleisch: Modularität als alternative Technologie- und Innovationsstrategie, Arbeitsberichte aus dem Fachbereich Informatik 21/2007

Kurt Lautenbach, Alexander Pinl: Probability Propagation Nets, Arbeitsberichte aus dem Fachbereich Informatik 20/2007

Rüdiger Grimm, Farid Mehr, Anastasia Meletiadou, Daniel Pähler, Ilka Uerz: SOA-Security, Arbeitsberichte aus dem Fachbereich Informatik 19/2007

Christoph Wernhard: Tableaux Between Proving, Projection and Compilation, Arbeitsberichte aus dem Fachbereich Informatik 18/2007

Ulrich Furbach, Claudia Obermaier: Knowledge Compilation for Description Logics, Arbeitsberichte aus dem Fachbereich Informatik 17/2007

Fernando Silva Parreiras, Steffen Staab, Andreas Winter: TwoUse: Integrating UML Models and OWL Ontologies, Arbeitsberichte aus dem Fachbereich Informatik 16/2007

Rüdiger Grimm, Anastasia Meletiadou: Rollenbasierte Zugriffskontrolle (RBAC) im Gesundheitswesen, Arbeitsberichte aud dem Fachbereich Informatik 15/2007

Ulrich Furbach, Jan Murray, Falk Schmidsberger, Frieder Stolzenburg: Hybrid Multiagent Systems with Timed Synchronization-Specification and Model Checking, Arbeitsberichte aus dem Fachbereich Informatik 14/2007

Björn Pelzer, Christoph Wernhard: System Description:“E-KRHyper“, Arbeitsberichte aus dem Fachbereich Informatik, 13/2007

Ulrich Furbach, Peter Baumgartner, Björn Pelzer: Hyper Tableaux with Equality, Arbeitsberichte aus dem Fachbereich Informatik, 12/2007

Ulrich Furbach, Markus Maron, Kevin Read: Location based Informationsystems, Arbeitsberichte aus dem Fachbereich Informatik, 11/2007

Philipp Schaer, Marco Thum: State-of-the-Art: Interaktion in erweiterten Realitäten, Arbeitsberichte aus dem Fachbereich Informatik, 10/2007

Ulrich Furbach, Claudia Obermaier: Applications of Automated Reasoning, Arbeitsberichte aus dem Fachbereich Informatik, 9/2007

Jürgen Ebert, Kerstin Falkowski: A First Proposal for an Overall Structure of an Enhanced Reality Framework, Arbeitsberichte aus dem Fachbereich Informatik, 8/2007

Lutz Priese, Frank Schmitt, Paul Lemke: Automatische See-Through Kalibrierung, Arbeitsberichte aus dem Fachbereich Informatik, 7/2007

Rüdiger Grimm, Robert Krimmer, Nils Meißner, Kai Reinhard, Melanie Volkamer, Marcel Weinand, Jörg Helbach: Security Requirements for Non-political Internet Voting, Arbeitsberichte aus dem Fachbereich Informatik, 6/2007

Daniel Bildhauer, Volker Riediger, Hannes Schwarz, Sascha Strauß, „grUML – Eine UML-basierte Modellierungssprache für T-Graphen“, Arbeitsberichte aus dem Fachbereich Informatik, 5/2007

Richard Arndt, Steffen Staab, Raphaël Troncy, Lynda Hardman: Adding Formal Semantics to MPEG-7: Designing a Well Founded Multimedia Ontology for the Web, Arbeitsberichte aus dem Fachbereich Informatik, 4/2007

Simon Schenk, Steffen Staab: Networked RDF Graphs, Arbeitsberichte aus dem Fachbereich Informatik, 3/2007

Rüdiger Grimm, Helge Hundacker, Anastasia Meletiadou: Anwendungsbeispiele für Kryptographie, Arbeitsberichte aus dem Fachbereich Informatik, 2/2007

Anastasia Meletiadou, J. Felix Hampe: Begriffsbestimmung und erwartete Trends im IT-Risk-Management, Arbeitsberichte aus dem Fachbereich Informatik, 1/2007

„Gelbe Reihe“
(http://www.uni-koblenz.de/fb4/publikationen/gelbereihe)

Lutz Priese: Some Examples of Semi-rational and Non-semi-rational DAG Languages. Extended Version, Fachberichte Informatik 3-2006

Kurt Lautenbach, Stephan Philippi, and Alexander Pinl: Bayesian Networks and Petri Nets, Fachberichte Informatik 2-2006

Rainer Gimnich and Andreas Winter: Workshop Software-Reengineering und Services, Fachberichte Informatik 1-2006

Kurt Lautenbach and Alexander Pinl: Probability Propagation in Petri Nets, Fachberichte Informatik 16-2005

Rainer Gimnich, Uwe Kaiser, and Andreas Winter: 2. Workshop "Reengineering Prozesse" – Software Migration, Fachberichte Informatik 15-2005

Jan Murray, Frieder Stolzenburg, and Toshiaki Arai: Hybrid State Machines with Timed Synchronization for Multi-Robot System Specification, Fachberichte Informatik 14-2005

Reinhold Letz: FTP 2005 – Fifth International Workshop on First-Order Theorem Proving, Fachberichte Informatik 13-2005

Bernhard Beckert: TABLEAUX 2005 – Position Papers and Tutorial Descriptions, Fachberichte Informatik 12-2005

Dietrich Paulus and Detlev Droege: Mixed-reality as a challenge to image understanding and artificial intelligence, Fachberichte Informatik 11-2005

Jürgen Sauer: 19. Workshop Planen, Scheduling und Konfigurieren / Entwerfen, Fachberichte Informatik 10-2005

Pascal Hitzler, Carsten Lutz, and Gerd Stumme: Foundational Aspects of Ontologies, Fachberichte Informatik 9-2005

Joachim Baumeister and Dietmar Seipel: Knowledge Engineering and Software Engineering, Fachberichte Informatik 8-2005

Benno Stein and Sven Meier zu Eißen: Proceedings of the Second International Workshop on Text-Based Information Retrieval, Fachberichte Informatik 7-2005

Andreas Winter and Jürgen Ebert: Metamodel-driven Service Interoperability, Fachberichte Informatik 6-2005

Joschka Boedecker, Norbert Michael Mayer, Masaki Ogino, Rodrigo da Silva Guerra, Masaaki Kikuchi, and Minoru Asada: Getting closer: How Simulation and Humanoid League can benefit from each other, Fachberichte Informatik 5-2005

Torsten Gipp and Jürgen Ebert: Web Engineering does profit from a Functional Approach, Fachberichte Informatik 4-2005

Oliver Obst, Anita Maas, and Joschka Boedecker: HTN Planning for Flexible Coordination Of Multiagent Team Behavior, Fachberichte Informatik 3-2005

Andreas von Hessling, Thomas Kleemann, and Alex Sinner: Semantic User Profiles and their Applications in a Mobile Environment, Fachberichte Informatik 2-2005

Heni Ben Amor and Achim Rettinger: Intelligent Exploration for Genetic Algorithms – Using Self-Organizing Maps in Evolutionary Computation, Fachberichte Informatik 1-2005