UNIVERSITÄT KOBLENZ · LANDAU

Hybrid State Machines with Timed Synchronization for Multi-Robot System Specification

Jan Murray, Frieder Stolzenburg, Toshiaki Arai

14/2005

Fachberichte INFORMATIK

ISSN 1860-4471

Universität Koblenz-Landau
Institut für Informatik, Universitätsstr. 1, D-56070 Koblenz

E-mail: researchreports@uni-koblenz.de,
WWW: http://www.uni-koblenz.de/FB4/

Hybrid State Machines with Timed Synchronization for Multi-Robot System Specification

Jan Murray[1], Frieder Stolzenburg[2], Toshiaki Arai[3]

[1] AI Research Group, Universität Koblenz-Landau, Universitätsstr. 1, D-56070 Koblenz, GERMANY, murray@uni-koblenz.de

[2] Automation and Computer Sciences Department, Hochschule Harz, Friedrichstr. 57-59, D-38855 Wernigerode, GERMANY, fstolzenburg@hs-harz.de

[3] Business Incubation Dept., Mitsubishi Materials Corporation, 1-297, Kitabukuro-cho, Omiya-ku, Saitama 330-8508, JAPAN, ara@mmc.co.jp

Abstract: In multi-robot systems such as in the RoboCup, the need for precise modeling or specification of agent behaviors arises due to the high complexity of the robot agent interactions and the dynamics of the environment. Since the behavior of agents usually can be understood as driven by external events and internal states, it is obvious to model multiagent systems by state transition diagrams. The corresponding formalisms come equipped with a formal semantics which is advantageous. In this paper, a combination of UML statecharts and hybrid automata is proposed, allowing formal system specification on different levels on abstraction on the one hand, and expressing real-time system behavior with continuous variables on the other hand. One important aspect of multi-robot systems is the need of coordination and hence synchronization of behavior. For both, statecharts and hybrid automata, usually it is assumed that synchronization takes zero time. This is sometimes unrealistic. Therefore, a new notation and implementation of synchronization is proposed here, which overcomes this problem. The proposed method is illustrated with a case study from the RoboCup domain. An example from an industrial application is also shown.

Keywords: cooperative robotics; robot behavior engineering; formal specification methods.

1 Introduction

In the field of multi-robot systems or physical multiagent systems the need for clear modeling or specification of agent behaviors arises due to the high complexity of their potential interactions and the dynamics of the environment. One important aspect of multi-robot systems is the cooperation of several robots to achieve a common goal. In this context it becomes clear that a central component of a multi-robot system specification should be a means of explicitly expressing *coordination* and *synchronization* of agents. Common means of modeling agent behaviors are based on various kinds of *state transition diagrams*. One main advantage of such a graphical notation is that it is intuitive and easy to understand. Another benefit is that many of these formalisms come equipped with a formal semantics, e.g. finite automata. This makes the specification accessible for formal methods in the agent design process. However, as physical agents act in an environment which allows for continuous processes, a means of expressing dependencies from continuous, quantitative data is desirable. The formalism of *hybrid automata* [5] is well suited for that.

The rest of the paper is organized as follows. In Section 2 hierarchical state machines are introduced, and Section 3 describes a way of incorporating hybrid automata into the formalism. Section 4 presents our formalism for expressing explicit synchronization among robots. A formal semantics for the proposed formalism is given in Section 5, while Section 6 presents an example from the RoboCup robotic soccer domain. Section 7 illustrates how industrial application can be modeled with this method. Some related work is presented in Section 8, and Section 9 finally concludes the paper.

2 Hierarchical State Machines

Statecharts are a part of UML [11, 12] and a well accepted means to specify dynamic behavior of software systems. The main concept for statecharts is a state, which corresponds to an activity or behavior of a robot agent. They can be described in a rigorously formal manner [13], allowing flexible specification, implementation and analysis of multiagent systems [8, 15] which is required for robot behavior engineering and modeling and simulating complex robots. Let us now define the underlying formal concepts of state machines, that are represented by statecharts.

Definition 1 (basic components) *The basic components of a* state machine *are the following four pairwise disjoint sets:*

- S: *a finite set of states, which is partitioned into three disjoint sets:* S_{simple}, $S_{composite}$ *and* $S_{concurrent}$ *— called simple, composite and concurrent states, containing one designated* start state $s_0 \in S_{composite}$,
- E: *a finite set of events,*
- X: *a finite set of (real-numbered) variables, and*
- A: *a finite set of actions.*

In statecharts, states are connected via *transitions* in $T \subseteq S \times S$ (represented as arrows), that are annotated with events in E, conditions and actions in A, indicating that an agent in the first state will enter the second state and perform specific *actions* when a specified *event* occurs and possibly additional conditions (called *guards*) are satisfied. Transitions are drawn as arrows labeled with an *annotation* of the form $e[g]/a$ where $e \in E$, $g \in G$ (the set of guards, i.e. first-order formulæ, including equations with variables), and $a \in A$. Since states may be simple, composite or concurrent, the behavior of agents or their state machines, respectively, cannot be described by sequences of simple states (as for plain finite state machines), but of configurations (see Def. 6. States are represented as rectangles with round corners and can be structured hierarchically. Following the lines of [11, 12], we define this structure as follows.

Definition 2 (state hierarchy) *Each state s is associated with zero, one or more* initial states $\alpha(s)$*: a simple state has zero, a composite state exactly one, and a concurrent state more than one initial states. Furthermore, each state $s \in S$ except s_0* belongs to *another state $\beta(s)$. $\beta(s)$ is defined for all $s \in S \backslash \{s_0\}$, and it must hold $\beta(s) \in S_{composite} \cup S_{concurrent}$. If $\beta(s) \in S_{concurrent}$, then $s \in S_{composite}$, i.e., a concurrent state must not be directly contained in another concurrent state. We assume that transitions keep to the hierarchy, i.e., if sTs' holds, then $\beta(s) = \beta(s')$.*

Example 1 *The state machine in Fig. 1 sketches the overall behavior of a robotic soccer agent. Some details are not shown, which is indicated by the* hidden decomposition icon ○—○*. The machine contains the simple states Init and GetBall, the composite states Behave, Defend, Marking, HandleBall, and Communicate, and the concurrent state Attack. Obviously, the start state is Behave, whose initial state is Init. The three states Init, Attack and Defend belong to the main (start) state Behave. Its initial state Init permits a transition annotated with KickOff/Kick(100%) (with empty guard that corresponds to True) to Attack, if the agent has a KickOff from the center point.*

3 Incorporating Hybrid Automata

Now, how can the behavior of state machines be described? Are they really adequate to model multiagent systems? – Usually we speak of *steps* of state machines. This suggests that we have a discrete model of time for multiagent systems. However, in practice, continuous activities should also be expressible, in order to model (physical) processes adequately. Hybrid Automata [5] are a formalism for describing such hybrid systems. They connect the modeling of discrete events by finite automata with the description of continuous activities by differential equations. Especially in the field of embedded systems the formalism is widely used.

Since hybrid automata are similar to statecharts, it makes sense to combine the advantages of both models. Statecharts have the clear advantage of allowing hierarchical specification on several levels of abstraction, while hybrid automata enable the introduction of continuous variables and flow conditions. Both automata models contain the concepts of states and transitions; we already defined this in the previous section. What is different, are the annotations at transitions (jump conditions) and the control conditions within one state (flow conditions and invariants). We will define this next.

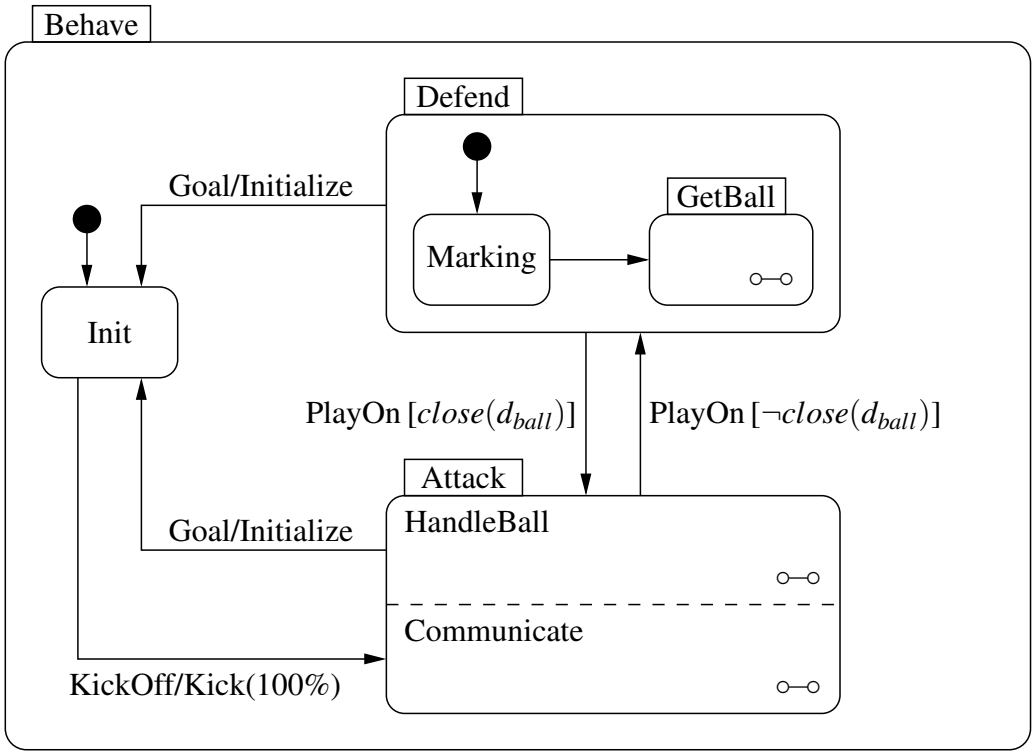

Figure 1: State machine for the overall behavior.

Definition 3 (jump conditions, flows and invariants) *In addition to the variables in X, we introduce new variables $\dot{x}$ (first derivatives during continuous change) and x' (values at the conclusion of discrete change) for each $x \in X$, calling the corresponding variable sets $\dot{X}$ and X', respectively. Then, each transition in T may be labeled by a* jump condition*, that is a predicate whose free variables are from $X \cup X'$. In addition, each state $s \in S$ is labeled with a* flow condition*, whose free variables are from $X \cup \dot{X}$, and an* invariant*, whose free variables are from X.*

The original definition of hybrid automata [5] focuses on flat automata only. But the combination with hierarchical statecharts, as done here, has the advantage of a layered architecture description on different levels of abstraction. The jump conditions from hybrid automata (Def. 3) make the explicit use of events and actions (see Def. 1) superfluous in principle. We will come back to this aspect in Section 5. But beforehand, we will discuss another important aspect of multi-robot systems, that can only partly be treated by state machines and hybrid automata, namely synchronization.

4 Synchronization

Usually the so-called *synchrony hypothesis* is adopted for state machines, assuming that the system is infinitely faster than the environment and thus the response to an external stimulus (event) is always generated in the same step that the stimulus is introduced. However in

practice, synchronization and coordination of actions cannot be done in zero time. In UML 1.5 [11], synchronization is present, but assumed to take zero time. In UML 2.0 [12] there does not seem to be a special synchronization mechanism available any longer except by join and fork transitions. Hence, it seems to be worthwhile considering synchronization in more detail. For this, we will introduce synchronization points which are associated with states, i.e. activities that last a certain time, and not with transitions (as in UML 1.5), because the transitions from one state to another one takes zero time according to the synchrony hypothesis.

Definition 4 (synchronization points) *A* synchronization point *(represented as oval) allows the coordinated treatment of common resources. It can be identified by special synchronization variables $x \in X_{synch} \subseteq X$ with a given maximal* capacity $C(x) \geq 0$. *Each such point may be connected with several states. We distinguish two relations: $R_+ \subseteq S \times X_{synch}$ and $R_- \subseteq X_{synch} \times S$, both represented by dashed arrows in the respective direction. Further, each connection in $R_+ \cup R_-$ is annotated with a number m with $0 \leq m \leq C(x)$.*

As just said, according to the previous definition, synchronization is connected to states and not to transitions as in UML 1.5. In consequence, it is now possible that synchronization may take some time as desired. The process of synchronization starts when a state s connected to a synchronization variable x is entered, and it ends only after some time when s is exited. Hence, we distinguish the allocation of (added or subtracted) resources and their (later) actual occupation by additional variables x_+ and x_- (used during the allocation phase) in each synchronization point. Hence, for each $x \in X_{synch}$, x_+ and x_- must be added to X.

Definition 5 (synchronization constraints) *Synchronization points impose additional constraints to the transitions incident with states s the synchronization variables x are connected to. For the outgoing edges, we distinguish* successful *and not* successful *synchronization. In the latter case, the transitions are marked with a crossed box ⊠. The following distinct cases have to be considered, where s' are states connected with s:*

1. *If sR_+x with annotation m, then (a) $x + x_+ + m \leq C(x)$ and $x'_+ = x_+ + m$ are added to all incoming transitions $s'Ts$, (b) $x'_+ = x_+ - m$ is added to all outgoing transitions sTs', and (c) further $x' = x + m$ is added to all successful outgoing transitions sTs'.*

2. *If xR_-s with annotation m, then (a) $x - (x_- + m) \geq 0$ and $x'_- = x_- + m$ are added to all incoming transitions $s'Ts$, (b) $x'_- = x_- - m$ is added to all outgoing transitions sTs', and (c) further $x' = x - m$ is added to all successful outgoing transitions sTs'.*

In [5], parallel composition of hybrid automata is introduced, which is related to synchronization. There, two (or more) automata may interact via joint events: if an event is an event in both automata, then they must synchronize on the respective transitions. This means, synchronization is represented rather implicitly in hybrid automata, whereas UML statecharts and the proposed combined notation stated here have concurrent states and regions for this purpose, which is more convenient. In both, UML statecharts and hybrid automata, it is assumed that synchronization takes zero time, which might not be the case in practice. Therefore it appears to be worthwhile to introduce explicit timed synchronization, as done here.

5 Configurations, Situations and State Change

Now, after having defined the syntax of hybrid statecharts with timed synchronization, we are ready to look at their semantics. Since states may be simple, composite or concurrent, the behavior of agents or their state machines, respectively, cannot be described by sequences of simple states (as for plain finite state machines), but of configurations (see also [2, 15]). Configurations are trees of states.

Definition 6 (configuration) *A configuration c is a rooted tree of states, where the root node is the topmost initial state of the overall state machine. A configuration must be completed by applying the following procedure as long as possible to leaf nodes:*

1. *If there is a leaf node in c labeled with a composite state s, then $\alpha(s)$ is introduced as immediate successor of s.*

2. *If there is a leaf node in c labeled with a concurrent state s, then all (composite) states s' with $\beta(s') = s$ become successor nodes of s.*

The state machine starts with the *initial configuration*, i.e. the completed topmost initial state of the overall state machine. In addition, an *initial condition* must be given, that is a predicate with free variables from $X \cup \dot{X}$. The current *situation* of the multiagent system is characterized by a pair (c, v) where c is a configuration and v is a valuation, i.e. a mapping $v : X \cup \dot{X} \to \mathbb{R}$. The *initial situation* at time $t = 0$ is a situation (c, v) where c is the initial configuration and v satisfies the initial condition.

The behavior can now be described by continuous and discrete state changes. Let (c, v) be the current situation, and $S(c)$ be the set of states occurring in c. As long as the conjunction of the invariants of all $s \in S(c)$ hold, the multiagent system evolves according to the flow conditions associated with all states $s \in S(c)$; we call this *continuous change*. Whenever after some time τ (chosen minimally) the invariants of one or more states do not hold any longer, then (and only then) a discrete state change takes place, i.e. a micro-step.

Definition 7 (micro-step) *A micro-step from one configuration c of a state machine to a configuration c' by means of a transition sTs' with some jump condition in the current situation (written $c \to c'$) is possible iff:*

1. *c contains a node labeled with s,*

2. *c' is identical with c except that s together with its subtree in c is replaced by the completion of s',*

3. *the jump condition of the given transition holds in the actual situation, and*

4. *the variables in X' are set according to the jump conditions.*

Fig. 2 shows some configurations of the state machine after several transitions for Ex. 1, showing only discrete change. Since there is a transition from Init to Attack with annotation

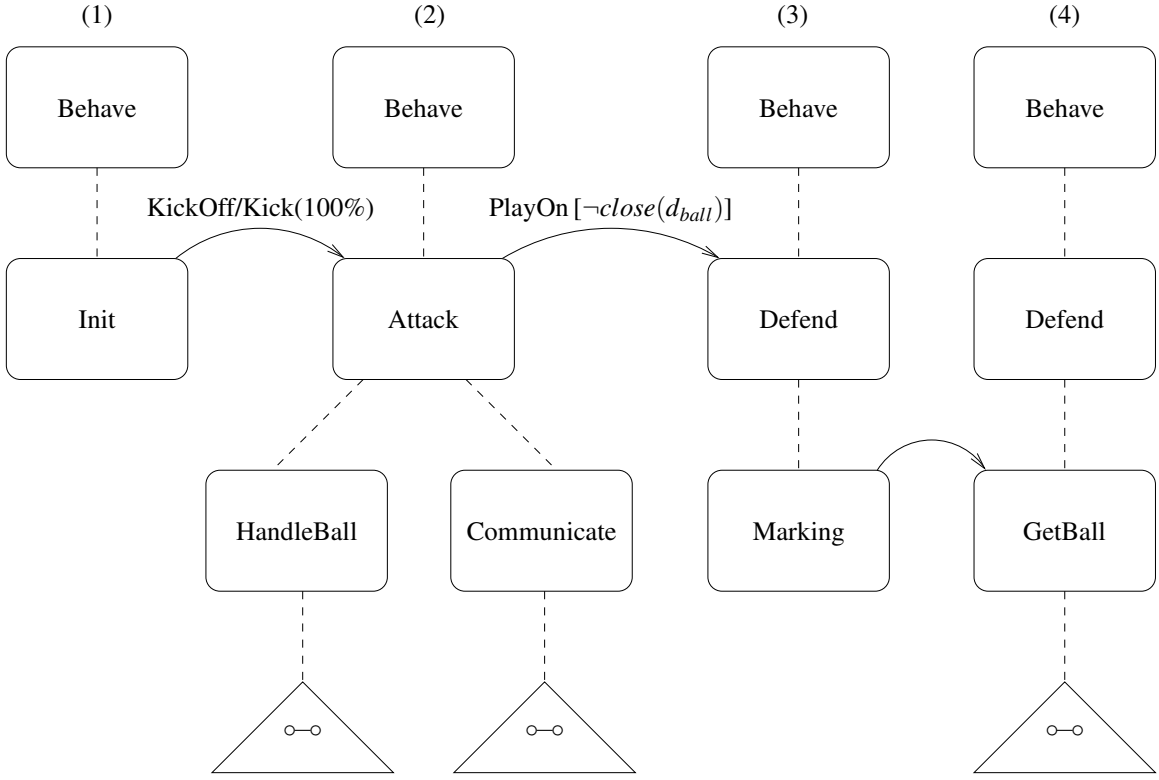

Figure 2: Configuration after several transitions.

KickOff/Kick(100%) in the state machine, the step from the first to the second configuration shown in Fig. 2 is possible according to Def. 7. For this, the Init node is replaced by Attack. Since Attack is a concurrent state, the (composite) states HandleBall and Communicate belonging to Attack, become successor nodes of Attack. Again, these states have to be completed. This is indicated by triangles with the symbol $\circ\!\!-\!\!\circ$ in it.

In the rare case, that after some time τ several invariants begin not to hold at the same time, then several micro-steps are performed in parallel for all respective states (called *macro-step* then). Conflicts may arise, if invariants of states on one and the same path are involved. In this case, the conflict must be solved. Here, outer transitions may be preferred over inner ones. The advantage of this procedure is that the agents are more reactive. In UML statecharts inner transitions have priority over outer transitions, while this is the other way round in [4]. In [7], priorities are not fixed, but they can be specified by the user.

Example 2 *Let us consider an example, shown in Fig. 3 (left). The initial condition at time* $t = 0$ *is* $x = 1$. *We start in the (only) state with flow condition* $\dot{x} = -x$, *i.e.* $x(t) = e^{-t}$. *The invariant* $x > 0.5$ *enforces a transition whenever* $x(t) = 0.5$, *i.e. after* $t' = \ln 2 \approx 0.693$. *There is a self-transition from this state back to itself with jump condition* $x' = 1$. *This means, that every* t' *seconds, the value of x begins to fall again (see Fig. 3, right).*

Note that we do not explicitly need the notion of events and actions here as in Def. 1. They are implicitly given by the conditions (Def. 3): events are stated by the jump conditions by the

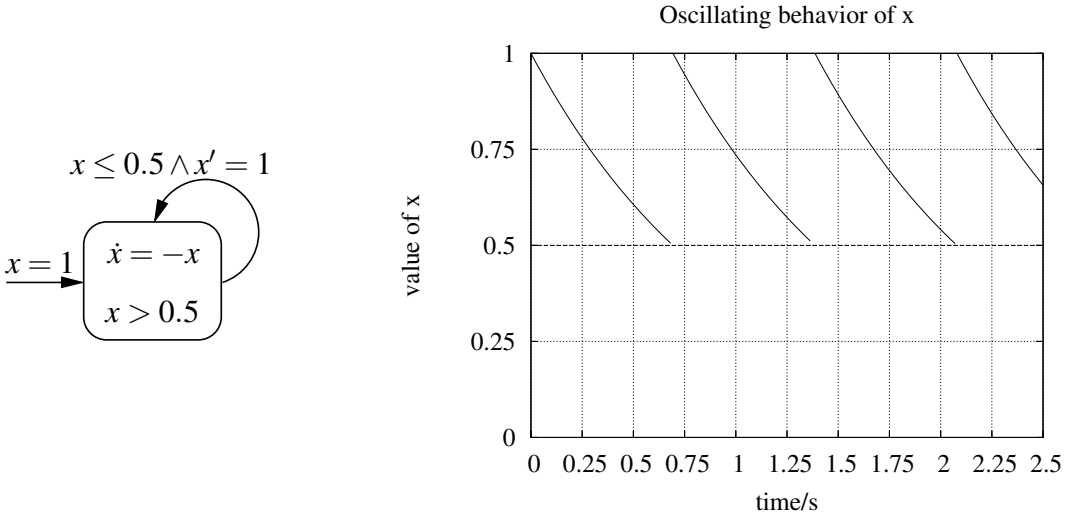

Figure 3: Example hybrid automaton. The diagram on the right shows the values of x as modeled by the hybrid automaton on the left.

expressions concerning the variables in X, and actions can be associated with the change of the variables in X' (see Def. 7, item 4.).

6 An Example from the RoboCup

Let us now consider a more elaborate example. In the RoboCup domain two teams of (possibly simulated) robotic agents engage in a match of soccer. Statechart based formalisms have already been used for specifying agent behaviors and multiagent plans in this domain [9, 8, 15].

One simple task the agents have to execute frequently is moving to the ball. In most situations it is advantageous if only one robot moves to the ball while the others position themselves strategically on the soccer field, e.g. for receiving a pass. Because of incomplete and noisy information this task needs explicit synchronization to avoid several agents going to the ball.

For this behavior, called `synchedGotoBall`, two roles have to be fulfilled, the *interceptor* (the agent going to the ball) and the *supporter*. The agent that is closer to the ball will take on the role of interceptor and execute the `gotoBall` behavior, while the supporter executes `strategicPosition`. If the supporter happens to come closer to the ball during the execution of the behavior, the agents exchange their roles, so that the former supporter now acts as interceptor and vice versa.

Example 3 (moving to the ball) *Fig. 4 shows the specification of a ball interception behavior for two agents in the RoboCup 3D simulation league with explicit synchronization. Note that the behavior specification is the same for interceptor and supporter. Instead of drawing two identical concurrent regions, one for each agent, we just add the cardinality n in a semi circle on the right of the diagram. This notation stands for n concurrent regions (states), where each*

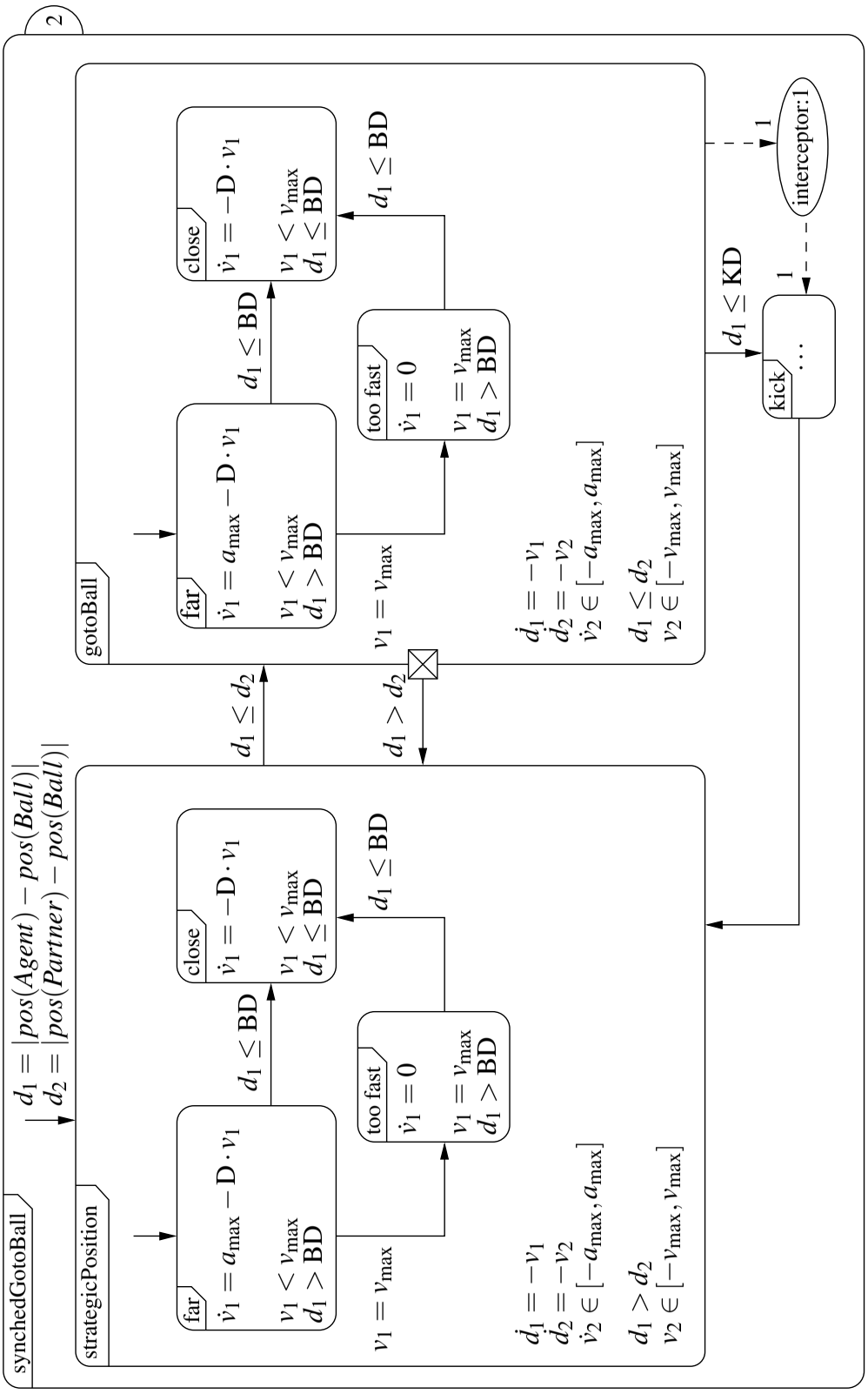

Figure 4: Specification of `synchedGotoBall` behavior by a hierarchical hybrid automaton. The synchronization point ensures that only one robot moves to the ball. The cardinality 2 shows that the behavior is executed by two homogeneous agents.

region gets new variables, i.e. all variables are local by default, except for the variables in X_{synch}, which are global variables.

Let us now take a more detailed look at the behavior of an agent. First the agent calculates the distance from the ball to himself (d_1) and to his partner (d_2). If he is closer to the ball, he takes the role of interceptor and executes `gotoBall`, otherwise he takes the role of supporter. As soon as the supporter is closer to the ball than the interceptor ($d_1 > d_2$) the agents swap their roles. Once the interceptor reaches the kicking distance (KD), i.e. he is close enough to kick the ball ($d_1 \leq \text{KD}$), he changes to the kicking behavior.

With the help of a synchronization point it is ensured, that only one agent moves to the ball, even if $d_1 = d_2$. To execute the `gotoBall` behavior an agent must be in possession of the *interceptor* resource. As this resource can only be held by one agent at a time, not both agents can move to the ball simultaneously. When the interceptor is close enough to kick the ball ($d_1 \leq \text{KD}$) he releases the resource. If `gotoBall` fails or the agents switch their roles, the resource is released automatically, so that the other agent is not blocked from going to the ball for ever.

Using differential equations and boolean expressions the execution of the behaviors can be described in greater detail. We can state general facts and constraints, e.g. that the distance to the ball decreases in proportion to the agent's velocity ($\dot{d_1} = -v_1$) or that during the execution of `strategicPosition` it must hold that $d_1 > d_2$.

The actual movement is modeled with the help of the states *far*, *close* and *too fast*. Note that the same sub automaton is used for modeling both `strategicPosition` and `gotoBall`. While the agent is far from the target position, he accelerates maximally ($a_{\max}$). Once he falls below the breaking distance (BD) the engine is switched off and the agent slows down due to friction ($-\text{D} \cdot v_1$). If the agent reaches the maximal velocity, the RoboCup simulator ignores any further acceleration. This is captured in the state named *too fast*. Note that the velocity remains constant in this state ($\dot{v}_1 = 0$).

7 Manufactory Process Example

In industrial applications, dynamic behavior of several agents can also be specified by means of the graphical UML notation. Timed synchronization as proposed in this article enables us to specify more realistic system behavior.

Example 4 *A generic process in computerized fabrication industry is modeled in Fig. 7, where a state transition for a certain part is described, in which it is manufactured and then inspected. After the inspection, if its condition is unsatisfiable, the part is returned to the previous sub-state, which is expressed by the crossed box. Subsequently the part is processed again. The synchronization point in Fig. 7 shows a constraint for all the parts; all of them must satisfy their post-condition, otherwise the final product should not be assembled. Other constraints may be applicable as well. For instance, in some processes, it is sufficient that at least 90% of elements meet certain value, i.e. not all parts need to satisfy the same criteria.*

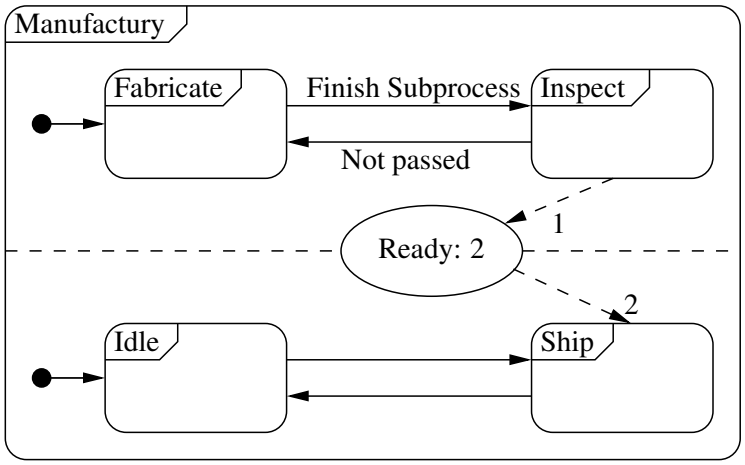

Figure 5: Manufactory process example.

As we see above, timed synchronization with the notation of synchronization points and constraints can be a powerful means in practice. Here, it allows us to express the fact that, whenever two parts are ready, they can be shipped by the respective unit.

8 Related Work

El Fallah et al. use hybrid automata for modeling the consumption of resources in multiagent plans [3]. A task is decomposed with the help of a global graph of functional dependencies, which also describes the relations among the different subtasks. Local plans for different agents are modeled with the help of hybrid automata. Synchronization of tasks is modeled by adding new states and labeled transitions to the automaton.

The extensible agent behavior specification language XABSL [6] uses XML to describe hierarchical state transition diagrams to specify agent behaviors in the RoboCup domain in the four-legged league. Which transitions are taken in a certain situation is described with the help of decision trees, that are also formulated with the help of XABSL. This formalism has strong relations to the procedure with (discrete) statecharts that already has been studied intensively in the RoboCup simulation league (see e.g. [9, 8]).

One way to integrate the formalisms of hierarchical state transition diagrams and hybrid automata has been proposed by Alur et al. [1] with the Charon language. But this approach is mainly used for specifying embedded systems. But there is no timed synchronization concept present as proposed here.

9 Conclusions

We presented a graphical formalism for specifying behaviors of multi-robot systems. With the help of hierarchical state transition diagrams behaviors of robots can be modeled conveniently, while the integration of hybrid automata facilitates the specification of continuous

processes. The use of concurrent regions allows for the modeling of behaviors that involve several physical agents. Explicit modeling of timed synchronization between several agents working together or sharing a resource can be realized with the help of synchronization points.

Future work includes devising means for the validation of specified behaviors with the help of temporal logics and model checking methods. In addition to that the notion of synchronization points has to be refined, e.g. to allow for modeling the exchange of data between agents. Finally, the proposed method will be applied to a greater extent in the RoboCup domain. First studies within the RoboCup 3D simulation league are encouraging.

Acknowledgments

This research has been carried out within the special research program DFG-SPP 1125 *Cooperative Teams of Mobile Robots in Dynamic Environments* under the grants *Fu 263/8* and *Sto 421/2*. This paper is a revised and extended version of [10].

References

[1] R. Alur, T. Dang, J. Esposito, Y. Hur, F. Ivancic, V. Kumar, I. Lee, P. Mishra, G. Pappas, and O. Sokolsky. Hierarchical modeling and analysis of embedded systems. In *Proceedings the 1st Workshop of Embedded Software*, 2001.

[2] T. Arai and F. Stolzenburg. Multiagent systems specification by UML statecharts aiming at intelligent manufacturing. In *Proceedings of the 1st International Joint Conference on Autonomous Agents & Multi-Agent Systems*, pages 11–18, Bologna, Italy, 2002. ACM Press. Volume 1.

[3] A. El Fallah-Seghrouchni, I. Degirmenciyan-Cartault, and F. Marc. Modelling, control and validation of multi-agent plans in dynamic context. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS04)*, pages 44–51, 2004.

[4] D. Harel and A. Naamad. The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, 1996.

[5] T. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS '96)*, pages 278–292, New Brunswick, New Jersey, 1996.

[6] M. Lötzsch, J. Bach, H.-D. Burkhard, and M. Jüngel. Designing agent behavior with the extensible agent behavior specification language XABSL. In Polani et al. [14], pages 114–124.

[7] G. Lüttgen, M. von der Beeck, and R. Cleaveland. A compositional approach to statecharts semantics. In *Proceedings of ACM SIGSOFT 8th International Symposium on*

the Foundations of Software Engineering, pages 120–129, San Diego, CA, 2000. ACM Press.

[8] J. Murray. Specifying agent behaviors with UML statecharts and StatEdit. In Polani et al. [14], pages 145–156.

[9] J. Murray, O. Obst, and F. Stolzenburg. Towards a logical approach for soccer agents engineering. In P. Stone, T. Balch, and G. Kraetzschmar, editors, *RoboCup 2000: Robot Soccer World Cup IV*, LNAI 2019, pages 199–208. Springer, Berlin, Heidelberg, New York, 2001.

[10] J. Murray and F. Stolzenburg. Hybrid state machines with timed synchronization for multi-robot system specification. In L. P. Reis, C. Carreto, E. Silva, and N. Lau, editors, *Proceedings of Workshop on Intelligent Robotics (IROBOT'2005)*, Covilhã, Portugal, 2005.

[11] Object Management Group, Inc. *OMG Unified Modeling Language Specification*, March 2003. Version 1.5.

[12] Object Management Group, Inc. *UML 2.0 Superstructure Specification*, October 2004.

[13] A. Pnueli and M. Shalev. What is in a step: On the semantics of statecharts. In T. Ito and A. R. Meyer, editors, *International Conference on Theoretical Aspects of Computer Software*, LNCS 526, pages 244–264, Sendai, Japan, 1991. Springer, Berlin, Heidelberg, New York.

[14] D. Polani, B. Browning, A. Bonarini, and K. Yoshida, editors. *RoboCup 2003: Robot Soccer World Cup VII*, LNAI 3020, Padua, Italy, 2004. Springer, Berlin, Heidelberg, New York.

[15] F. Stolzenburg and T. Arai. From the specification of multiagent systems by statecharts to their formal analysis by model checking: Towards safety-critical applications. In M. Schillo, M. Klusch, J. Müller, and H. Tianfield, editors, *Proceedings of the 1st German Conference on Multiagent System Technologies*, LNAI 2831, pages 131–143, Erfurt, 2003. Springer, Berlin, Heidelberg, New York.

Available Research Reports (since 2000):

2005

14/2005 *Jan Murray, Frieder Stolzenburg, Toshiaki Arai.* Hybrid State Machines with Timed Synchronization for Multi-Robot System Specification.

13/2005 *Reinhold Letz.* FTP 2005 — Fifth International Workshop on First-Order Theorem Proving.

12/2005 *Bernhard Beckert.* TABLEAUX 2005 — Position Papers and Tutorial Descriptions.

11/2005 *Dietrich Paulus, Detlev Droege.* Mixed-reality as a challenge to image understanding and artificial intelligence.

10/2005 *Jürgen Sauer.* 19. Workshop Planen, Scheduling und Konfigurieren / Entwerfen.

9/2005 *Pascal Hitzler, Carsten Lutz, Gerd Stumme.* Foundational Aspects of Ontologies.

8/2005 *Joachim Baumeister, Dietmar Seipel.* Knowledge Engineering and Software Engineering.

7/2005 *Benno Stein, Sven Meier zu Eißen.* Proceedings of the Second International Workshop on Text-Based Information Retrieval.

6/2005 *Andreas Winter, Jürgen Ebert.* Metamodel-driven Service Interoperability.

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

4/2005 *Torsten Gipp, Jürgen Ebert.* Web Engineering does profit from a Functional Approach.

3/2005 *Oliver Obst, Anita Maas, Joschka Boedecker.* HTN Planning for Flexible Coordination Of Multiagent Team Behavior.

2/2005 *Andreas von Hessling, Thomas Kleemann, Alex Sinner.* Semantic User Profiles and their Applications in a Mobile Environment.

1/2005 *Heni Ben Amor, Achim Rettinger.* Intelligent Exploration for Genetic Algorithms – Using Self-Organizing Maps in Evolutionary Computation.

2004

12/2004 *Manfred Rosendahl.* Objektorientierte Implementierung einer Constraint basierten geometrischen Modellierung.

11/2004 *Urs Kuhlmann, Harry Sneed, Andreas Winter.* Workshop Reengineering Prozesse (RePro 2004) — Fallstudien, Methoden, Vorgehen, Werkzeuge.

10/2004 *Bernhard Beckert, Gerd Beuster.* Formal Specification of Security-relevant Properties of User-Interfaces.

9/2004 *Bernhard Beckert, Martin Giese, Elmar Habermalz, Reiner Hähnle, Andreas Roth, Philipp Rümmer, Steffen Schlager.* Taclets: A New Paradigm for Constructing Interactive Theorem Provers.

8/2004 *Achim Rettinger.* Learning from Recorded Games: A Scoring Policy for Simulated Soccer Agents.

7/2004 *Oliver Obst, Markus Rollmann.* Spark — A Generic Simulator for Physical Multi-agent Simulations.

6/2004 *Frank Dylla, Alexander Ferrein, Gerhard Lakemeyer, Jan Murray, Oliver Obst, Thomas Röfer, Frieder Stolzenburg, Ubbo Visser, Thomas Wagner.* Towards a League-Independent Qualitative Soccer Theory for RoboCup.

5/2004 *Peter Baumgartner, Ulrich Furbach, Margret Groß-Hardt, Thomas Kleemann.* Model Based Deduction for Database Schema Reasoning.

4/2004 *Lutz Priese.* A Note on Recognizable Sets of Unranked and Unordered Trees.

3/2004 *Lutz Priese.* Petri Net DAG Languages and Regular Tree Languages with Synchronization.

2/2004 *Ulrich Furbach, Margret Groß-Hardt, Bernd Thomas, Tobias Weller, Alexander Wolf.* Issues Management: Erkennen und Beherrschen von kommunikativen Risiken und Chancen.

1/2004 *Andreas Winter, Carlo Simon.* Exchanging Business Process Models with GXL.

2003

18/2003 *Kurt Lautenbach.* Duality of Marked Place/Transition Nets.

17/2003 *Frieder Stolzenburg, Jan Murray, Karsten Sturm.* Multiagent Matching Algorithms With and Without Coach.

16/2003 *Peter Baumgartner, Paul A. Cairns, Michael Kohlhase, Erica Melis (Eds.).* Knowledge Representation and Automated Reasoning for E-Learning Systems.

15/2003 *Peter Baumgartner, Ulrich Furbach, Margret Gross-Hardt, Thomas Kleemann, Christoph Wernhard.* KRHyper Inside — Model Based Deduction in Applications.

14/2003 *Christoph Wernhard.* System Description: KRHyper.

13/2003 *Peter Baumgartner, Ulrich Furbach, Margret Gross-Hardt, Alex Sinner.* 'Living Book' :- 'Deduction', 'Slicing', 'Interaction'..

12/2003 *Heni Ben Amor, Oliver Obst, Jan Murray.* Fast, Neat and Under Control: Inverse Steering Behaviors for Physical Autonomous Agents.

11/2003 *Gerd Beuster, Thomas Kleemann, Bernd Thomas.* MIA - A Multi-Agent Location Based Information Systems for Mobile Users in 3G Networks.

10/2003 *Gerd Beuster, Ulrich Furbach, Margret Groß-Hardt, Bernd Thomas.* Automatic Classification for the Identification of Relationships in a Metadata Repository.

9/2003 *Nicholas Kushmerick, Bernd Thomas.* Adaptive information extraction: Core technologies for information agents.

8/2003 *Bernd Thomas.* Bottom-Up Learning of Logic Programs for Information Extraction from Hypertext Documents.

7/2003 *Ulrich Furbach.* AI - A Multiple Book Review.

6/2003 *Peter Baumgartner, Ulrich Furbach, Margret Groß-Hardt.* Living Books.

5/2003 *Oliver Obst.* Using Model-Based Diagnosis to Build Hypotheses about Spatial Environments.

4/2003 *Daniel Lohmann, Jürgen Ebert.* A Generalization of the Hyperspace Approach Using Meta-Models.

3/2003 *Marco Kögler, Oliver Obst.* Simulation League: The Next Generation.

2/2003 *Peter Baumgartner, Margret Groß-Hardt, Alex Sinner.* Living Book – Deduction, Slicing and Interaction.

1/2003 *Peter Baumgartner, Cesare Tinelli.* The Model Evolution Calculus.

2002

12/2002 *Kurt Lautenbach.* Logical Reasoning and Petri Nets.

11/2002 *Margret Groß-Hardt.* Processing of Concept Based Queries for XML Data.

10/2002 *Hanno Binder, Jérôme Diebold, Tobias Feldmann, Andreas Kern, David Polock, Dennis Reif, Stephan Schmidt, Frank Schmitt, Dieter Zöbel.* Fahrassistenzsystem zur Unterstützung beim Rückwärtsfahren mit einachsigen Gespannen.

9/2002 *Jürgen Ebert, Bernt Kullbach, Franz Lehner.* 4. Workshop Software Reengineering (Bad Honnef, 29./30. April 2002).

8/2002 *Richard C. Holt, Andreas Winter, Jingwei Wu.* Towards a Common Query Language for Reverse Engineering.

7/2002 *Jürgen Ebert, Bernt Kullbach, Volker Riediger, Andreas Winter.* GUPRO – Generic Understanding of Programs, An Overview.

6/2002 *Margret Groß-Hardt.* Concept based querying of semistructured data.

5/2002 *Anna Simon, Marianne Valerius.* User Requirements – Lessons Learned from a Computer Science Course.

4/2002 *Frieder Stolzenburg, Oliver Obst, Jan Murray.* Qualitative Velocity and Ball Interception.

3/2002 *Peter Baumgartner.* A First-Order Logic Davis-Putnam-Logemann-Loveland Procedure.

2/2002 *Peter Baumgartner, Ulrich Furbach.* Automated Deduction Techniques for the Management of Personalized Documents.

1/2002 *Jürgen Ebert, Bernt Kullbach, Franz Lehner.* 3. Workshop Software Reengineering (Bad Honnef, 10./11. Mai 2001).

2001

13/2001 *Annette Pook.* Schlussbericht "FUN - Funkunterrichtsnetzwerk".

12/2001 *Toshiaki Arai, Frieder Stolzenburg.* Multiagent Systems Specification by UML Statecharts Aiming at Intelligent Manufacturing.

11/2001 *Kurt Lautenbach.* Reproducibility of the Empty Marking.

10/2001 *Jan Murray.* Specifying Agents with UML in Robotic Soccer.

9/2001 *Andreas Winter.* Exchanging Graphs with GXL.

8/2001 *Marianne Valerius, Anna Simon.* Slicing Book Technology — eine neue Technik für eine neue Lehre?.

7/2001 *Bernt Kullbach, Volker Riediger.* Folding: An Approach to Enable Program Understanding of Preprocessed Languages.

6/2001 *Frieder Stolzenburg.* From the Specification of Multiagent Systems by Statecharts to their Formal Analysis by Model Checking.

5/2001 *Oliver Obst.* Specifying Rational Agents with Statecharts and Utility Functions.

4/2001 *Torsten Gipp, Jürgen Ebert.* Conceptual Modelling and Web Site Generation using Graph Technology.

3/2001 *Carlos I. Chesñevar, Jürgen Dix, Frieder Stolzenburg, Guillermo R. Simari.* Relating Defeasible and Normal Logic Programming through Transformation Properties.

2/2001 *Carola Lange, Harry M. Sneed, Andreas Winter.* Applying GUPRO to GEOS – A Case Study.

1/2001 *Pascal von Hutten, Stephan Philippi.* Modelling a concurrent ray-tracing algorithm using object-oriented Petri-Nets.

2000

8/2000 *Jürgen Ebert, Bernt Kullbach, Franz Lehner (Hrsg.).* 2. Workshop Software Reengineering (Bad Honnef, 11./12. Mai 2000).

7/2000 *Stephan Philippi.* AWPN 2000 - 7. Workshop Algorithmen und Werkzeuge für Petrinetze, Koblenz, 02.-03. Oktober 2000 .

6/2000 *Jan Murray, Oliver Obst, Frieder Stolzenburg.* Towards a Logical Approach for Soccer Agents Engineering.

5/2000 *Peter Baumgartner, Hantao Zhang (Eds.).* FTP 2000 – Third International Workshop on First-Order Theorem Proving, St Andrews, Scotland, July 2000.

4/2000 *Frieder Stolzenburg, Alejandro J. García, Carlos I. Chesñevar, Guillermo R. Simari.* Introducing Generalized Specificity in Logic Programming.

3/2000 *Ingar Uhe, Manfred Rosendahl.* Specification of Symbols and Implementation of Their Constraints in JKogge.

2/2000 *Peter Baumgartner, Fabio Massacci.* The Taming of the (X)OR.

1/2000 *Richard C. Holt, Andreas Winter, Andy Schürr.* GXL: Towards a Standard Exchange Format.