

Specifying Rational Agents with Statecharts and Utility Functions

Oliver Obst*

Universität Koblenz-Landau, AI research group
Rheinau 1, D-56075 Koblenz, GERMANY
`fruit@uni-koblenz.de`

Abstract. To aid the development of the robotic soccer simulation league team RoboLog-2000, a method for the specification of multi-agent teams by statecharts has been introduced. The results in the last years competitions showed that though the team was competitive, it did not behave adaptive in unknown situations. The design of adaptive agents with this method is possible, but not in a straightforward manner. The purpose of this paper is to extend the approach by a more adaptive action selection mechanism and to facilitate a more explicit representation of goals of an agent.

1 Introduction

Creating a team of autonomous agents for dynamic environments like simulated robotic soccer is a difficult task, because agents can encounter a number of different situations the designer might not have been thinking of at the time of creation. Statecharts [3] are a visual approach to formalize complex systems. They can be used to describe the behavior of a system on different levels of granularity with one representation. The specification can be transformed in a straightforward manner into running Prolog code [7, 5]. The method facilitates a high degree of reactivity, since only a few rules have to be checked each cycle. As a proof of concept, we developed the RoboLog 2000 simulator team using our approach [6]. During the competitions in Amsterdam (EuRoboCup 2000) and Melbourne (RoboCup 2000), it turned out that our team performed quite well in many situations. On the other hand, in new situations our team did not behave adaptive enough. This became clear against new teams *crossing* the ball in front of our goal and in matches with a high loss or a high win. Designing adaptive teams with our approach would have been possible by intensively using hierarchically structured statecharts [12]. On the other hand, this was not straightforward, and the goals of an agent were determined implicitly by its actions. The purpose of this paper is to extend our approach by a more adaptive action selection mechanism and to facilitate a more explicit representation of goals of an agent.

* This research is supported by the grant *Fu 263/6-1* from the German research foundation *DFG*.

The idea of our new approach is as follows: We want to specify multi-agent scripts with statecharts, so agents are using scripts as parameterized, predefined plans. An agent should select which of the applicable scripts is most useful for its current goals. The goals of an agent are in its set of beliefs and can be changed by a script. The action selection mechanism becomes more adaptive, because the predefined plans are chosen more carefully for a new situation. Though a second step towards more adaptiveness is the alteration of predefined plans, we focus on the sooner aspect in this paper.

Additionally, we want to provide a mechanism to evaluate the utility of a script that takes the commitment to selected options into account. Like in our former approach, there should be a possibility to generate the agent program from its specification.

1.1 A First Approach to Multi-Agent Systems with Statecharts

In [7], we introduced a method for specifying multi-agent behavior by statecharts and the translation of statecharts into code. For the design of soccer agents, the statecharts we are using are a natural way of describing a sequence of actions for a certain situation: While designing agents, the designer might think of typical situations for a certain player and some appropriate actions during this situation. Situations and appropriate actions can be formalized with the statecharts. Once the agent design is finished, it consists of a number of scripts.

At the time of execution, the agent looks for an applicable script and executes a step of the first one it can find. Because of the way actions are specified in this approach, representation of goals of our agents is implicit in the agent's behavior specification: if the preconditions for an action hold, the action is executed. The conditions are regarded in the order of specification, that is, most of the time, more specific rules are regarded first. If no specific rule is applicable, as a last resort a default rule is applied. As long as a specific rule holds, this mechanism ensures that the script the agent is going to execute is applicable. What it does not guarantee is that the script which is going to be used is the best or most useful one.

1.2 Action Selection

A rational agent is one that tries to maximize its performance measure [10], that is in our case the expected success of its actions. Until now, the measure for expected success were boolean conditions on edges of a statechart. Actions at an edge with applicable conditions were expected to be successful. If an agent had more than one option to execute actions, the first applicable one was selected. A more adaptive action selection mechanism should evaluate the usefulness of applicable options and execute the most useful one. As an intended side effect of an improved option evaluation and explicit goal representation, our agent model is closer to the BDI model [9].

2 Utilities in Scripts

In [13], an architecture for action selection based on preferences and the probability of success is introduced. In this architecture, options can be scored and executed. The score of an option depends on a preference (or reward) and on the probability for success of that option. Both the reward and the probability values can be hand written or are subject to reinforcement learning. The option evaluation described in their paper is used for the player with the ball only. It is a single decision what to do with the ball: the player decides what to do, executes it and during the next simulation step, the situation is evaluated anew.

Their approach supports the construction of reactive agents that behave locally optimal with respect to their information. However, the disadvantage is that building complex plans consisting of several steps or including some team mates can only happen implicitly by intelligently selecting preference values when building the agent.

For an option selection mechanism that can be used for all players in a team throughout the whole match, it is important to consider how long an agent remains committed to an option. There are situations where it is an advantage to continue a locally non-optimal script, since we are specifying the behavior of a multi-agent system. If agents do not switch behavior due to only small changes in the environment, agents can rely on each other. As Kinny [4] found out, agents with strong commitment to their plans behave superior to cautious agents that always reconsider their plans, even in rapidly changing environments.

2.1 Preferences and Probabilities in Statecharts

Basically, a statechart without preferences consists of the four finite and pairwise disjoint sets S , E , G and A , where S is a set of states, E is a set of events, G is a set of guard conditions and A is a set of actions. Usually, a transition from a state s to state s' is labeled with an annotation of the form $e[g]/a$, where e is from the set of events E , g from the set of guards G and a from the set of actions A . The (informal) meaning of an annotation like this is that if the agent is in state s and an event e occurs, the agent should execute action a and find itself in state s' afterwards, provided that condition g holds.

Additionally, it is possible to structure statecharts hierarchically. Within hierarchically structured statecharts, a transition from one state to the next is possible, if both states belong to the same state(chart). A configuration within such a statechart is a rooted tree of states [12]. A step from one configuration to the next can happen exactly once every simulation step. After a transition, the agent saves the current script and configuration. In the simulated soccer environment, the agent returns control to the soccer server and waits until the next simulation cycle starts. After the belief update for the next cycle, the agent tries to continue the script with the saved configuration.

For our new approach, we need to change the statecharts so that the agent can estimate the preference value for an action annotated at a transition. The preference for a script is clearly dependent on the current belief (in the BDI

sense) of the state of the environment. The second important parameter for the preference for a script is the current state of the statechart that should be executed. Usually, this is going to be the initial state of a statechart, since each script should be progressed from its beginning. Only the current executed script might be in a different state. So for each script, the preference function maps the agent's belief and the state of the statechart to a value, the preference value of a script. If we want to regard the uncertainty whether a script will or will not be successful, the preference for a script and the probability for success make up the utility of a script.

Instead of events and actions, transitions are now labeled with annotations $p[g]/a$, where p is a preference function returning a value evaluating the preference for the annotated action, given the agents' belief of the environment. A preference function consists of one or more *attribute*, the conditions in the former approach.

Definition 1 (Attribute function). *Let S be a set of states of a statechart, A a set of actions, and Bel the current belief of the agent about the state of the world. An attribute α is a function $\alpha : S \times A \times Bel \mapsto [0..1]$.*

The preference of an agent for a certain action is given by the set of attributes at a transition from one state to the next. Like in [13], the difficulty lies in the selection of appropriate values for different options. Preferences have to be comparable, so that options with a higher score turn out to be a better decision for the agent.

For most actions, it will be useful to combine some attributes to a preference, for instance the preference value of a pass could depend on the possibility to kick the ball, the distance to the team mate and the positions of the opponents. For our purposes, we make the simplifying assumption that all attributes are independent of each other. The advantage of this approach is that we can use an multiplicative value function for conjunctions.

Definition 2 (Preference Function).

1. *If α is an attribute function and $w \in \mathbb{R}$, $w \cdot \alpha$ is a preference function. w is the weight of the preference function. w_{max} is the maximum preference for all preference functions.*
2. *If p_1 and p_2 are preference functions, $p_1 \wedge p_2$ is a preference function.*
3. *If p_1 and p_2 are preference functions, $p_1 \vee p_2$ is a preference function.*
4. *If p is a preference function, $\neg p$ is a preference function.*

For a conjunction, the value of a preference of an action is the product of values of all weighted attributes at the transition: $P = \prod_{i=0}^{n-1} w_i \cdot \alpha_i$, where α_i are the attributes and w_i are the weights. For disjunctions of attributes, we use the maximum value of all attributes in the disjunction: $P = \max w_i \cdot \alpha_i \Big|_0^{n-1}$. The preference value of a negated preference function $\neg p$ is w_{max} minus the value of p .

Additionally, it is possible to use probabilities whether an action will succeed. Besides the preferences, the probabilities are a measure for the utility of an

action. The probability whether an action will succeed can be estimated by statistics collected over a series of matches against other teams. The probability of an action is a value $k \in [0..1]$ that is multiplied with the preference value for the respective action.

2.2 Commitment

As said earlier, most of the the time it is an advantage if agents continue their plans once started instead of switching behaviors all the time. However, there are still situations where an agent should stop its current plan and restart the script evaluation. For instance, a statechart for double passing could have a global outgoing edge that should be selected if the play mode changes. In this case, the selection of another statechart was intended at the time the agent was developed. So the agent has to evaluate all alternatives emitting from its current state, those continuing the statechart and those leading to a terminal state. In the latter case, the configuration of the statechart is reset. Additionally, at every step the agent has to evaluate the utilities of some other actions, namely those annotated at edges emitting from initial states of statecharts different from the current one. If the agent finds an action from another statechart to be more useful, it resets the configuration of the current statechart to the initial state and starts to progress the transitions of the new statechart beginning from its initial state.

In order to reflect the stronger commitment to an already selected script, utilities of scripts with a state unequal to the initial state are multiplied by a commitment factor c ($c \geq 1$). The commitment factor is a global value, so it is easy to adjust the tendency of the agent to switch behaviors.

There is yet another case to look at, namely circular edges. Circular edges might be useful for specifying repetition, for instance successive dashes or waiting for a pass. However, the utility of a script could decrease if a certain step in a script is executed a number of times. In case of the pass, it is certainly going to be the case that after some steps of unsuccessful waiting to receive the ball the usefulness of a longer wait is low. In other cases, the number of repetitions does not have any influence on the utility of a script, for instance when dashing to a certain position. To provide means to distinguish different passes of execution of a statechart, both the state s and the number of the current pass p are used to estimate the utility of a script. For each state s in a statechart \mathcal{S} there is a value $d_s \in [0..1]$ which is used to decrease the utility of an action exponentially by the number of passes through a state. s_0 denotes the initial state of a statechart, while p_0 stands for the first pass through a state. The commitment of an agent to an action is defined by the following equation.

$$C_{\mathcal{S}}(s, p) = \begin{cases} 1, & \text{if } s = s_0 \text{ and } p = p_0 \\ c \cdot d_s^p, & \text{otherwise} \end{cases} \quad (1)$$

Altogether, the following parameters are used to estimate the utility of an action: k the probability of success for an action, the according preference function as the product of all its attributes and $C_{\mathcal{S}}(s, p)$, the commitment to selected

options given the current state s and the current pass p . For the utility function, see the equation below.

$$U_S(s, p) = k \cdot C_S(s, p) \cdot \prod_{i=0}^{n-1} w_i \cdot \alpha_i \quad (2)$$

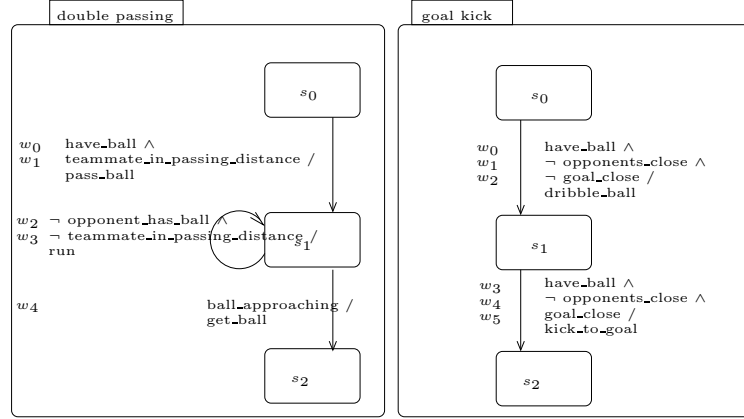


Fig. 1. Statecharts with preferences

If the utility of an action in the current statechart is lower than the utility of some applicable action in another statechart, state s and pass p are reset to s_0 and p_0 respectively.

In Fig. 1, there is a (simplified) example for statecharts specifying two kinds of behavior of the agent. The designer annotates the transitions with some attributes and actions. Additionally, weights have to be specified. If both statecharts are in state s_0 , the agent program evaluates the preference for each transition from state s_0 to state s_1 and executes the annotated action, i.e. `pass_ball` or `dribble_ball`, in this case.

Specification of team behavior is possible in the same manner: the transitions between states of a script performed by some agents are annotated with the preference functions for the roles of the respective agent. A description of a situation where some agents work together is given by the configuration of the collaborating agents and the state of the environment (following the description in [12]). Transitions within the same statechart describe the behavior of participating agents, where each agent selects actions according to its preferences. In situations where teamwork is useful, the preference functions assign a higher value to the behaviors in a multiagent script. To resolve possible mismatches concerning the current state of environment, communication can be used.

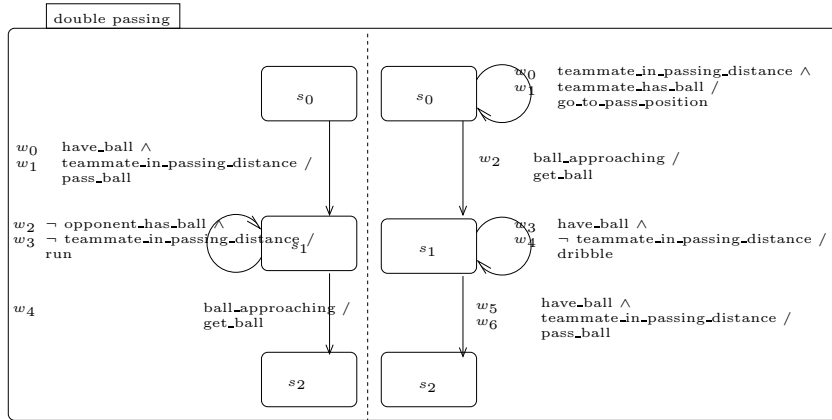


Fig. 2. Statechart for two agents

3 Translating Statecharts with Preferences into Code

One of the advantages of our former approach was, that the statecharts could be translated easily into Prolog code. But what has to be done to translate statecharts with utility functions into running code? Assuming the system provides the attribute functions defined in the previous section and an interface to the actions of an agent, the agent designer specifies statecharts for typical situations. Additionally, the designer combines attribute functions and weights to utility functions and annotates transitions with appropriate utility/action pairs. It is clearly more complicated to find the right conditions *plus* the appropriate values for the weights. As in [13], it is possible to leave this calibration to some learning procedure. On the other hand, simulation of our former approach is possible when using only preference values of 0 and 1 as truth values. From this point of view, it is possible to adjust the values “by hand” and to do a refinement later on.

To translate the statecharts into running code, the system has to be equipped with some control loop processing the resulting scripts. For an outline of the agent control loop, have a look at Fig. 3.

To actually map the relevant part of belief for an action to some real value, we were assuming that the system provides attribute functions doing this. Dependent on the type of information, continuous or binary functions have to be used. An attribute encoding the fact, that the player has the ball could be binary (0 or 1), whereas an attribute saying that the team mate is in passing distance could employ the sigmoid function, as described in [13].

For each statechart, the translation procedure takes the transitions from one state to the next and produces code like in Fig. 4. Unlike in the former approach,

```

1: while true do
2:   update belief
3:   if the guard condition at the current transition does not hold then
4:     reset state and pass of current script
5:   end if
6:   calculate utility for all scripts
7:   make script with the highest utility the current script
8:   execute a step of current script
9: end while

```

Fig. 3. Agent Control Loop

now there are two types of rules for each transition: the first one for estimating the utility of an action and the second one for actually executing the action.

```

transition(pass_ball, s0, s1, Utility) :-
    have_ball(U0),
    closest_teammate(No),
    in_passing_distance(No,U1),
    Utility is w0 · U0 · w1 · U1.

transition(pass_ball, s0, s1) :-
    closest_teammate(No),
    teammate_dist(No,Dist),
    teammate_dir(No,Dir),
    power_for_dist(Dist, Power),
    kick_ball(Power, Dir).

```

Fig. 4. Prolog code for a transition in a statechart

4 Related Work

Scerri and Ydrén [11] work on an approach providing means to create complex agent systems for the end user. End users are domain experts, but non-programmers. The high level behavior of agents in their approach can be edited using a graphical tool. Diagrams the user creates are translated into a behavior based system, where different behaviors run in parallel according to their activation function. End users cannot change the lower levels of the behavior, in contrast to our approach, where the specification method can be used for low level and high level behavior specification. The user interface in [11] is tailored for the soccer domain and the user can specify the behavior directly on the soccer field.

Dorer [2] tackles the problem of action selection by extending behavior networks to continuous domains. Goals of an agent activate competence modules

assigned to the goals and, if the value of activation and executability for the most activated module lies above some threshold, the associated behavior is executed.

In Dix et al. [1], the authors present a general framework for dealing with uncertainty in agents. They provide different semantics for probabilistic agent programs based on deontic logic of actions. They define an agent as a set of probabilistic rules under which an agent is obliged, permitted, or forbidden to take some actions. However, they do not address the problem of selecting the best or most useful action.

Poole [8] introduces the independent choice logic (ICL), a framework for specifying decision problems in multi-agent systems. The approach is very expressive and tries to combine advantages of Markov decision process, influence diagrams and logic programming.

5 Conclusions

In this paper we introduced statecharts with utilities as a method to specify autonomous agents. The specification of an agent can be translated into a running program automatically. The resulting agent program is both reactive and committed to actions it once started. Compared to our previous approach, we improved the adaptiveness of our agents.

For our future work, we want to investigate the use of learning methods to estimate preference values and weights. Offline learning could aid the developer during the design process. Especially online learning seems to be challenging, as this would – again – contribute to adaptive behavior.

References

1. Jürgen Dix, Mirco Nanni, and V. S. Subrahmanian. Probabilistic Agent Programs. *ACM Transactions of Computational Logic*, 1(2):207–245, 2000.
2. Klaus Dorer. Behavior networks for continuous domains using situation-dependent motivations. In *International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 1233–1238, 1999.
3. David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.
4. David Kinny and Michael P. Georgeff. Commitment and effectiveness of situated agents. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, Sydney, Australia, 1991.
5. Jan Murray. Soccer agents think in UML. Master's thesis, Universität Koblenz-Landau, Fachbereich Informatik, 2001.
6. Jan Murray, Oliver Obst, and Frieder Stolzenburg. RoboLog Koblenz 2000. In Peter Stone, Tucker Balch, and Gerhard Kraetzschmar, editors, *RoboCup-2000: Robot Soccer World Cup IV*, Lecture Notes in Artificial Intelligence, pages 469–472. Springer, Berlin, Heidelberg, New York, 2001. Team description.
7. Jan Murray, Oliver Obst, and Frieder Stolzenburg. Towards a logical approach for soccer agents engineering. In Peter Stone, Tucker Balch, and Gerhard Kraetzschmar, editors, *RoboCup-2000: Robot Soccer World Cup IV*, Lecture Notes in Artificial Intelligence, pages 199–208. Springer, Berlin, Heidelberg, New York, 2001.

8. David Poole. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94(1-2):5–56, 1997. Special Issue on Economic Principles of Multi-agent Systems.
9. Anand S. Rao and Michael P. Georgeff. Modeling rational agents within a BDI-architecture. Technical Note 14, Australian Artificial Intelligence Institute, February 1991.
10. Stuart Russell and Peter Norvig. *Artificial Intelligence. A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ, 1995.
11. Paul Scerri and Johan Ydrén. End user specification of RoboCup teams. In *RoboCup-99: Robot Soccer World Cup III*, number 1856 in LNAI, pages 450–459. Springer, 2000.
12. Frieder Stolzenburg. Reasoning about cognitive robotics systems. In Reinhard Moratz and Bernhard Nebel, editors, *Themenkolloquium Kognitive Robotik und Raumrepräsentation des DFG-Schwerpunktprogramms Raumkognition*, Hamburg, 2001.
13. Peter Stone and David McAllester. An architecture for action selection in robotic soccer. In *Submitted to the Fifth International Conference on Autonomous Agents*, 2001.