From Sketch To Plan

Stephan Gspandl and Michael Reip and Gerald Steinbauer and Franz Wotawa
Institute for Software Technology, Graz University of Technology
Inffeldgasse 16B/II, 8010 Graz, Austria

Abstract

One of the most important tasks in the RoboCup soccer leagues is to build up soccer domain knowledge for decision-making. Therefore, most RoboCup teams adapt tactics graphs to develop their agents’ behavior manually. This is a very laborious and time-consuming job.

In this paper, we propose a methodology for an easy translation of soccer tactics sketches to plans defining the agents’ roles and abstract behaviors in different situations.

Our research focus thus comprises the questions how such a system can be designed, how the transformation of tactics sketches can be implemented and evaluated as well. First experiments in the RoboCup Soccer Simulation League show promising results in using tactics sketches to define the agents’ behavior.

Introduction

In all RoboCup soccer leagues it is very important to build up soccer domain knowledge. In most teams this is a time-consuming and laborious job of transforming soccer tactics (esp. from books like (Bangsbo and Peitersen 2000) or (Lucchesi 2002)) or other concepts into specific models, plans or source code. This causes many hybrid and incompatible descriptions of the same ideas making exchange and mutual improvements virtually impossible.

Our Simulation League team KickOffTUG for example is very complex to manage in this respect. The rule base is very large, containing over 200 actions and conditions for a single agent. It takes at least one person to put constant effort in definition and adaption of the hand-maintained plans.

Same applies for many other teams in this league.

We believe adapting the agents’ behavior should be as simple, fast and intuitive as sketching on a piece of paper or scanning diagrams. It would save a tremendous amount of time, if for example soccer tactics graphs could be automatically transformed into plans the agents could execute. In order to create a team good enough to participate in championships we simply might scan soccer literature. Using a few sketches explaining how to exploit specific weaknesses of other teams this team might actually outperform other teams.

Or imagine a logistics robot which should be trained to delivery parts from the warehouse to a newly installed assembly line. You may either program it to perform this task or simply use a touch-sensitive display and sketch the desired behavior.

Therefore, we are constantly working on a methodology for an easy translation of dynamic sketches (i.e. sketches that not only contain spatial information but also the chronological order of strokes) into behavioral descriptions. Fig. 1 shows an example of a tactics graph with two strikers which are expected to break the opposing line of defense. The first striker should dribble towards the field center along the line of defense and then pass into the penalty box to its teammate. In the meantime this teammate gets around the opposing player and into the penalty field where it can accept the pass. This graph is a good example how books on soccer tactics encode the desired behavior. As simple as this tactic looks a lot of human knowledge and intuition is necessary to interpret it correctly: The sequences of actions have to be performed in a way that a meaningful synchronized interaction is created.

Humans intuitively recognize relevant and irrelevant facts, features and relations of this scene, i.e. that

- striker 1 is marked
- striker 1 is unable to pass by the opponent line of defense
- striker 1 is able to dribble along the line of defense
- the position from where striker 1 should pass is a good one to do so
- ...

All these pieces of information are implicit qualitative descriptions of the scene making the understanding of it possible. For humans to understand this scene, it is not interesting to know that striker 2 is exactly 1.5 meters outside...
the penalty box. Although it might be discussed whether the given tactic is meant to be performed only outside the penalty box, a human striker who has internalized the idea depicted can act accordingly regardless of exact positions of the actors.

These facts have to be considered when trying to automate the transformation of a sketch to action. Our proposed methodology comprises several stages of transformation and models of representation. The input is a static graph (a scan) or a dynamic sequence of strokes depicting an agent's behavior in a concrete situation. The output will be a description on what the agents should do in the depicted situation. Several of these descriptions therefore yield a model suitable for decision-making. This means, when faced with a situation an agent can look-up what to do next.

In the following section we discuss related research. In the succeeding section we present the proposed methodology in more details. This section is followed by experiments and empirical results and we conclude with a discussion and future research.

**Related Research**

The methodology proposed is influenced by mainly two groups which have developed several methods in this field: the group around Randall Davis at the MIT Artificial Intelligence Laboratory as well as Kenneth D. Forbus of the Qualitative Reasoning Group at Northwestern University. Concerning research at the MIT, we can identify many relevant topics like early processing, object interpretation, graphical description of behavior, sketch recognition and learning and describing symbols. They developed several well-known applications (tools for the design of two-dimensional machines as in (Alvarado 2000), to transform UML-sketches into code as in (Hammond and Davis 2002) or a CAD application as in (Lipson and Shpitalni 1997) at the Mechanical Engineering Department). All projects share the underlying description language LADDER (Hammond and Davis 2003). In this language basic components (symbols or glyphs) to be recognized are described using constraints. These constraints don’t have to be created manually, but instead constraint features can be learned on the base of human perception models (Veselova and Davis 2004). Furthermore, stroke-ordering information is applied to make recognition even more robust (Sezgin and Davis 2007).

The Qualitative Reasoning Group on the other hand pursues a different approach. They focus on annotations to provide an open-domain tool (nuSketch) for sketch recognition, based on the assumption that recognizing glyphs (as they call basic components) correctly is not essential (see (Forbus et al. 2004)). They base interpretation on these annotations by applying logic axioms specific to a certain domain. To define such a sketching domain, they utilize a knowledge base (a subset of Cyc; see http://www.opencyc.org/) from which these relevant facts and relations are selected (using sKEA, the sketching Knowledge Entry Associate). (Forbus and Usher 2002)

We believe a fundamental recognition of symbols is inevitable for reasons of generality and intuition. We don’t want to sacrifice the naturalness of sketching by expecting annotations. A robust, general methodology should essentially be able to recognize and interpret sketches without meta-information. Merely when no correct interpretation can be inferred or when adding new symbols, annotations are used to support and enhance the process. Merely to support and enhance the process and when adding new symbols, annotations should be provided, combining the approaches of both groups.

In contrast to the research at the MIT, we will further concentrate on methods to automatically interpret arbitrary sketches on the base of a domain model. We will extend the discussed transformations and models of representation to derive a general model of arbitrary sketched intention. Each piece of information (like a new stroke drawn) is used to infer new conclusions and may also threaten existing. In terms of soccer someone might, for example, want to sketch the situation of a pass in front of the goal, i.e. draws an agent and an arrow towards the goal. The system might at this stage reason that the agent tries to score a goal. As new information becomes available, i.e. a second agent with an arrow indicating that it runs to the goal, it is evident that the first arrow represents a pass.

**Transformation Process**

The process transforming a given sketch into a behavioral model comprises several stages and is depicted in Fig. 2. In the first two stages the tactics graph should be recognized and translated to a quantitative spatial-temporal representation of the described situation. This model contains a set of primitives, such as circles, boxes, ... as well as strokes. In the RoboCup domain, for example, a set of circles typically represents the own, the opposing players and the ball and a set of different arrows the actions to be performed by the players (see Fig. 3). By using a domain model the spatial-temporal model is transformed into a qualitative domain-specific situation model. This model in turn comprises a set of literals describing the situation given by the strategy graph, the actions which are applied to this situation and a set of conditions describing the situation after application of

![Image: Figure 2: The process of automatically deriving a behavioral model from several tactics graphs or sketches.](image-url)
STRIPS

A general planning problem is typically described by the problem of correctly deriving heuristics of reward and time expenditure as necessary for T-R Programs can be avoided. The action to be executed is retrieved from the tree by depth-first searching for an executable action the effects of which are not yet fulfilled (see also (Gspandl et al. 2007), and (Gspandl et al. 2008)). With respect to the STRIPS notation actions are described as a extended tuple \( \{ P, C_{\text{pre}}, C_{\text{inv}}, C_{\text{post}}, A, P_A \} \) with

1. \( P \) a set of parameters to the node
2. \( C_{\text{pre}} \) a set of preconditions
3. \( C_{\text{inv}} \) a set of invariants
4. \( C_{\text{post}} \) a set of postconditions
5. \( A \) the identifier of an implemented skill or a Plan Tree \( A' \)
6. \( P_A \) a set of parameters which are passed to \( A' \) at execution

Pre- and postconditions resemble preconditions and effects of the STRIPS notation, i.e. tell the decision-making routine whether the skill or Plan Tree \( A' \) can be called or a state described by the node has already been reached respectively. Invariants are an additional set of conditions, which help deciding whether the last executed skill should be pursued further or a better alternative should be searched for. This adds to balancing between deliberativity and reactivity which is essential in very dynamic environments.

Each T-R Program is inserted into the tree upside-down. The root (a null action) depicts the general start of the traversing algorithm, each leaf a goal of one T-R Program and siblings alternative sequences of behavior. As a single T-R Program only represents one goal of an agent, the tree consists of all goals.

Consider the following example: An agent should find the overall model of the agents’ behavior.

![Diagram of objects](image)

Figure 3: Objects which have to be recognized in the formalism of (Bangsbo and Peitersen 2000).

**Representation of Behavioral Plans**

The output of the transformation process is a plan describing the agents’ behaviors in different situations. The plan model we have implemented for the decision-making of our Simulation League team KickOffTUG is used as general model for this description. The underlying formalism uses the common representations of the Stanford Research Institute Problem Solver (STRIPS) (Nilsson and Fikes 1971) as well as the Teleo-Reactive Program (TR) formalism (Nilsson 1994).

STRIPS A general planning problem is typically described by an initial state, an anticipated goal and a set of actions to reach this goal. The Closed-World Assumption provided, only positive literals can be used to describe each state of the world. The STRIPS notation uses the following distinction. States and goals are represented by conjunctions of positive literals, whereas actions (or operators (Nilsson and Fikes 1971)) are specified by (1) name and parameters of the action, (2) its preconditions and (3) its effects.

Concrete examples for STRIPS-based planning concepts are state-space search, partial-order planning as well as plan graphs (Ghallab, Nau, and Traverso 2004).

Teleo-Reactive Programs Nilsson introduced the term ‘Teleo-Reactive Program’ or short ‘T-R Program’. T-R Programs combine deliberativity with reactivity and are represented by a sequence of actions with associated conditions:

\[
C_1 \rightarrow A_1 \\
C_2 \rightarrow A_2 \\
... \\
C_m \rightarrow A_m
\]

Each condition \( C_i \) describes the state to be given in order that the related action \( A_i \) is executable. An action might be a primitive action or a T-R Program again. The topmost action (which is a null action) of a T-R Program represents the goal via its conditions. Starting at this root the program is searched downward step by step for a satisfied condition. The action associated with the first condition to be found can then be executed.

As actions are not necessarily discrete, the action is continuously executed as long as its condition remains the topmost satisfied condition. If the state changes in a way that the actual action can no longer be performed or a superior action is now executable, a more basic action or an action which is closer to fulfilling the goal is chosen respectively. Thus, conditions also have to be evaluated continuously.

Semantically spoken, each action in a T-R Program is responsible for establishing the state necessary for the execution of the next higher action in the program. This fundamental property is called Regression Property.

Additionally, T-R Programs have to be complete, i.e. the disjunction of all conditions in a single T-R Program has to be a tautology. At least one action within a T-R Program has thus to be executable. T-R Programs which satisfy both properties are called universal.

Basically, T-R Programs are linear. If more than one action exists fulfilling a certain condition, these can be combined in a tree. A T-R Tree is traversed by searching the shallowest node with conditions satisfied. (Nilsson 1994)

Benson et al. elaborate how autonomous agents can be implemented using this T-R formalism (Benson and Nilsson 1995). In order to choose the best-possible T-R Program which should be executed in a given situation (i.e. to settle on the goal which the agent should try to reach), they use heuristics of execution time and reward. Thus, the goal is pursued which yields the highest reward as fast as possible.

**Plan Trees**

For the RoboCup Soccer Simulation League team KickOffTUG we have introduced a formalism based on the concepts of the STRIPS notation (Nilsson and Fikes 1971) as well as T-R Programs (Nilsson 1994), where all single T-R Programs are merged into one tree (which will be further called Plan Tree). The T-R Programs are weighted by the order in which they are attached to the tree. Left nodes are explicitly prioritized over siblings. Thereby, the problem of correctly deriving heuristics of reward and time expenditure as necessary for T-R Programs can be avoided. The action to be executed is retrieved from the tree by depth-first searching for an executable action the effects of which are not yet fulfilled (see also (Gspandl et al. 2007), and (Gspandl et al. 2008)).

With respect to the STRIPS notation actions are described as a extended tuple \( \{ P, C_{\text{pre}}, C_{\text{inv}}, C_{\text{post}}, A, P_A \} \) with

1. \( P \) a set of parameters to the node
2. \( C_{\text{pre}} \) a set of preconditions
3. \( C_{\text{inv}} \) a set of invariants
4. \( C_{\text{post}} \) a set of postconditions
5. \( A \) the identifier of an implemented skill or a Plan Tree \( A' \)
6. \( P_A \) a set of parameters which are passed to \( A' \) at execution

Pre- and postconditions resemble preconditions and effects of the STRIPS notation, i.e. tell the decision-making routine whether the skill or Plan Tree \( A' \) can be called or a state described by the node has already been reached respectively. Invariants are an additional set of conditions, which help deciding whether the last executed skill should be pursued further or a better alternative should be searched for. This adds to balancing between deliberativity and reactivity which is essential in very dynamic environments.

Each T-R Program is inserted into the tree upside-down. The root (a null action) depicts the general start of the traversing algorithm, each leaf a goal of one T-R Program and siblings alternative sequences of behavior. As a single T-R Program only represents one goal of an agent, the tree consists of all goals.

Consider the following example: An agent should find the
ball, run to it and then try to score a goal or make a pass. This would yield two T-R Programs:

\[
\begin{align*}
\text{canScore} & \rightarrow \text{scoreGoal} \\
\text{seeBall} & \rightarrow \text{runToBall} \\
\text{true} & \rightarrow \text{findBall}
\end{align*}
\]

\[
\begin{align*}
\text{canPass} & \rightarrow \text{pass} \\
\text{seeBall} & \rightarrow \text{runToBall} \\
\text{true} & \rightarrow \text{findBall}
\end{align*}
\]

These two plans can now be merged into one Plan Tree. The resulting tree is shown in Fig. 4. It has only one branching point, where the agent has to decide whether to score or to pass. As scoring is always more important (if possible) than passing, this action is prioritized in the tree. If the agent is for example currently in ball possession but the way to the goal is blocked, he will therefore decide to pass.

Figure 4: An exemplary Plan Tree of two simple T-R Programs.

**Stages of the Transformation Process**

Three main stages perform the transformation from input strokes to a behavioral model: Sketch Recognition, Sketch Interpretation and Combination. In order to elaborate the ideas, Fig. 1 is used as simple running example.

**Sketch Recognition** Sketch Recognition transforms a sketch or scan into a domain-specific quantitative spatial-temporal model. This stage is divided into two steps: First, primitives are recognized domain-independently to provide a preprocessing. In the second step a set of rules is applied to attach meaning to each basic entity.

Formally, the first transformation is an application of LADDER-rules to a given set of strokes. The second transformation is based on the recognized primitives and the original set of strokes. A set of domain-specific recognition rules is applied to create the quantitative spatial-temporal model of the sketch. For example, a circle is mapped to an own agent and several arrows to actions to be performed (see Fig. 3).

Let’s use the tactics graph in Fig. 1 as example. Recognizing this sketch would yield a set of facts describing the depicted situation quantitatively:

\[
\begin{align*}
\text{agent(a1)} \\
\text{own(a1)} \\
\text{pos(a1, 28.07, -13.04)}
\end{align*}
\]

\[
\begin{align*}
\text{field(f1, 52.5, -34, 52.5, 34,} \\
-52.5, 34, -52.5, -34) \\
\text{//field is defined by its corners}
\end{align*}
\]

\[
\begin{align*}
\text{action(di, dribble,} \\
28.07, -13.04, 28.71, -3.59) \\
\text{//action has a type plus start/end}
\end{align*}
\]

\[
\ldots
\]

**Sketch Interpretation** Sketch Interpretation is a transformation from a purely quantitative model to a qualitative description of the sketch in terms of the Plan Tree formalism. A set of qualitative rules is applied to infer meaning. The idea is that not the exact position of two strokes is important. Instead the spatial relationships between drawn entities are more relevant. For example, consider the situation where we have two circles and a line with an arrow in a sketch. An arrow pointing from one circle to the other has a meaning. Whereas stating that the circles, the line, and the arrow have a certain position from the lower left corner of the sketch is less relevant. Hence, interpretation relies on qualitative descriptions of drawn entities stating their relationships from which we derive logical models.

The application of a qualitative domain model thus yields a set of positive literals which are then employed to create a Plan Tree. In our example this set could contain predicates from the decision-making domain of our team KickOffTUG like:

\[
\begin{align*}
\text{• defensive(opponent1)} \land \\
\text{defensive(opponent2)} \land \\
\text{defensive(opponent3)} \land \\
\text{offensive(own1)} \land \\
\text{offensive(own2)} \land \\
\text{pos(own1, outside_penalty_left)} \land \\
\text{...}
\end{align*}
\]

\[
\begin{align*}
\text{• hasBall(own1)} \land \\
\text{dribbleFreeSector(own1, right)} \land \\
\text{marking(opponent1, own1)} \land \\
\text{...}
\end{align*}
\]

In order to formally transform this description into a Plan Tree, the following algorithm is applied:

1. The possible action templates have to be defined, i.e. a set of basic action preconditions, a set of action invariants and a set of axioms describing how the execution influences the world. For example, a basic precondition for a pass would be ball possession, an invariant that the ball is moving from pass start to its target and an action axiom states that after the execution the pass partner is now in ball possession.

2. The set of positive literals as shown in the previous paragraph is employed as preconditions to the new Plan Tree.

3. For each action specified by the sketch:

   The action preconditions are the postconditions of the previous action except for the first action where the preconditions are equal to the plan preconditions.

   The postconditions are derived by applying the predefined action axioms to the preconditions (progression).

   The action invariants have to be set according to the action template.

   The action is inserted into the tree according to the temporal sequence. There is a separate branch for each actor in the tree. For the given example, one branch is
4. The plan invariant has to be derived from all action invariants. It is the conjunction of invariants of each branch. The invariant of a branch is the disjunction of all action invariants in the branch. For the running example the second striker ensures via this plan invariant that he is still able to break the opposing line of defense and the first striker is still committed to the mutual plan, i.e. either dribbling or passing as described.

Figure 5: The plan resulting from the transformation of the tactics graph in Fig. 1.

The Plan Tree resulting from the transformation of the running example is shown in Fig. 5. It abstracts the interaction between two players A and B according to the tactic. During execution, i.e. when an agent needs to come to a decision, the actual environment is inspected whether it matches the plan’s preconditions. If so, this means that there are two players which satisfy the situation’s constraints. Thus, the placeholders A and B are substituted with the concrete identifiers, say playerOne (some teammate) and self (the player in need of a decision). For self it has to be checked whether the action run can be executed. If so, the agent performs this action.

In the subsequent cycle, the same action run is inspected again. The action’s invariants have to be satisfied (Is performing run still reasonable?) as well as the plan’ invariants (Is every participating player still committed to the plan?). After assuring that playerOne is still dribbling or (as it might be the case now) is passing, it continues with its action. By iteratively applying this routine the agents deliberately approach their common goal as long as the environment remains supportive. If something occurs which prevents that the goal can be reached successfully, the agents react immediately and derive a new decision.

Sketch Combination Finally, multiple models have to be transformed into a general model. This can be done by simply inserting all Plan Trees under a common root. As each situation broadens the tree further, this will probably turn out to be very inefficient for large decision spaces. Therefore, techniques to deepen the tree have to be employed. One approach would be binary ramification, i.e. using common predicates as anchor points to ramify the tree.

Experiments

In order to answer the question whether an automation of sketch interpretation can be achieved satisfactorily, we implemented the presented methodology in the context of the RoboCup Soccer Simulation League. A MATLAB recognition engine preprocesses tactics graphs or static sketches for interpretation. The interpretation is based on the KickOffTUG domain model (a set of predicates relevant for soccer). This domain was built to evaluate current situations a soccer agent might encounter. By applying this set of predicates to the spatial-temporal model of the recognition yields a simple plan which can be executed by the soccer agents. A series of experiments has been conducted to answer the questions

- whether this methodology can be useful in describing behaviors
- an existing domain model can be applied to perform the third stage ‘Interpretation’ of the sketching process
- what the performance of a system designed using the proposed methodology is

Implementation

In the following sections the current implementation of each stage is presented as foundation for the discussion.

Recognition At the moment, the recognition engine works directly on scanned tactics graphs. Fig. 3 shows the basic components identified by the image recognition module. Similar versions of this representation can be found throughout literature. Meta-information about positions and directions is derived from the middle circle. Output of the recognition is a spatial-temporal description of the symbols in XML.

Deriving a Plan Tree from a Sketch The input to the interpretation can either be a drawn or scanned static sketch preprocessed by the recognition described above, but also a situation put together graphically with a few clicks. Both provide entities already recognized for the interpretation. These tools are only the first step to a fully-integrated sketching system.

Thus, a domain-specific spatial-temporal model with players and the ball recognized and annotated with field positions as well as actions to be performed is handed over to the interpretation. The interpretation is based on the KickOffTUG soccer domain model. This is a set of predicates implemented to evaluate situations a soccer agent could encounter. Examples would be the predicates hasBall(A), checking whether an agent A is in ball possession, or passPossible(X, Y), examining whether agent X can safely pass to agent Y. Applying this as domain yields a qualitative description of a given tactics situation. Simply spoken, this description is currently derived by executing the predicate evaluation functions with all reasonable parameter combinations. The combination of all positive literals provides the preconditions of the depicted tactics situation. The Plan Tree is then created according to the algorithm described above.

Experiments Set-Up

We have conducted three sets of experiments to test various aspects of our proposed methodology. First, on a set of 12 randomly created plans coverage and quality is measured. In a second experiment, these plans are compared against
hand-maintained plans of the team KickOffTUG. Last, a set of 5 transformed sketches is created and evaluated whether they can improve the performance of a weak team.

**Coverage and Quality** In order to evaluate coverage and quality of the plans resulting from our implementation, we create a random set of sketches out of logfiles (i.e. recorded matches) and annotate it with intention (compare Fig. 6). Thus, we can ensure that the depicted situations can be abstracted robustly without risking conforming only to a certain tactics book.

Figure 6: An exemplary, randomly generated situation (Plan 11 in Fig. 7 and 8) which is used as input to the experiments.

In our first two experiments we further only consider goal-preparing or offensive situations as these are of particular interest. Such situations are defined with the ball near the opponent goal and our team in ball possession. From each sequence of possible situations in 10 logfiles, we draw three by chance. The resulting set serves as ‘knowledge input’ to our interpretation. Again each situation already comprises a spatial-temporal model for the soccer domain. Therefore, there is no need for recognition. This does not influence the general applicability of the approach.

Next, the set of predicates of our team serves as grounding domain for interpretation. Each condition is evaluated over all combinations of entities of each situation and each entity and the result stored as preconditions to a new plan, and thus yields a set of plans.

This set of plans is the test set which is evaluated over 15 test games in the following way: The decision-making is presented with each offensive situation from the logs. Thus, the set of plans is searched for those plans whose preconditions are true for the given situation. This is called a hit. The set of plans is then sorted by hits and similar plans are resolved heuristically to ensure the quality of the plans.

The resulting set of plans is reviewed in two ways over 15 further test games:

1. The plan coverage is calculated by checking how many offensive situations can be covered with various selections from the plan set.
2. The plan quality is evaluated by comparing whether the hitting situations resemble the original sketch from which the plan was created.

**Comparison to a team with good performance** In a second set of experiments, the plan quality is determined statistically by simulating a tournament of 100 games. The 12 derived plans from the first experiment are connected to our hand-maintained standard plan (the plan our agents make decisions on in tournaments) to the left on top-level. Thus, they are examined first when searching for a decision and override any other decision in the existing plan whenever they match. Furthermore, an appropriate action has been added to each plan. This means, whenever a plan hits, the according action is executed. The better the situation has been modeled, the better the influence of the action should be on the score. On this base, 50 games have been simulated with sketch plans and 50 without.

The logfiles of these games are then reviewed offline. Each set of 50 games is recalculated. Whenever a hit occurs, the logs are checked whether the offensive situation as described by the plan has had (for simulations with sketch plans) or would have had (for simulations without sketch plans) an influence on the score. On both sets the ratio between the number situations resulting in a goal and the overall count of hit situations is calculated and compared. We can thus see, how the test plans performed in the modeled situations against what to expect from the standard plans.

**Comparison to a simple team** In the third set of experiments, a set of 5 plans is created from soccer books in the same way as above. As our implementation is not yet able to handle interaction, the tactics are rather simple. The behavior of a simple team has been defined by hand in order to evaluate whether the performance of these simple hand-crafted plans can be enhanced by the proposed methodology. Thus, the enhanced simple team is created by adding the 5 plans to the base plans as in the last experiment.

Again, a tournament has been simulated. This time, there have been 25 games with the simple team and 25 with the enhanced team against a common opponent team. To compare these teams’ performance, the scores have been reviewed.

**Results**

Results of our tests showed that we are indeed able to describe soccer situations in 2D RoboCup Simulation League automatically with our proposed methodology. In order to provide deployable plans for the Simulation League team, manual post-processing of the plans or adaption of the model are yet necessary in this setup. Fig. 7 shows the accumulated plan coverage for twelve plans derived by the previously described process and sorted by their extent of contribution. In total it is possible to achieve a coverage of 30.3
percent of all offensive situations with only 12 plans. Concerning quality of the plans, we have tested the resemblance of hitting situations to the original. Experts have rated whether a situation was resembled correctly or not. Fig. 8 depicts their ratings on the twelve plans. In total 43.69 percent of situations were resembled correctly. To further increase classification results a dedicated domain model for recognition and interpretation is certainly necessary. Results of the second experimental setup illustrate the quality by simulation. The results of the experiments convey the number of situations that contributed to a goal in comparison to the overall number of situations that occurred over 50 games, simulated with and without sketch plans. An average of 39 percent compared to 59 percent combined with the manual evaluation of situations conveys that the modeling was successful for most cases, but still could be improved. We assume that this lower number is caused by the fact that only about 44 percent of the situations could be resembled correctly. This value will be used as characteristic value to rate further improvements.

The performance of the simple team over the sketch-enhanced team is illustrated in the average goals scored per game. The simple team scored an average of 2.65 whereas the enhanced team was able to score an average 2.8 goals. The minor difference of 0.15 goals between the teams can be explained by the simple team. The tactics described in soccer books can only be applied to teams with at least average performance. A team not able to build up a decent attack will hardly create situations depicted by offensive soccer tactics. Consider Fig. 1 as example: a team of agents running randomly to the ball will never end up in a situation where a dribbling towards the center and a pass into the penalty will be possible. Similar applies for the enhanced simple team. Thus, we can conclude that these sketches can account for better performance, but not until a team has reached average performance, it can really profit from soccer tactics.

**Conclusion**

We have presented a methodology for transforming tactics sketches to behavioral models and its current implementation on base of the RoboCup Soccer Simulation League. A series of experiments was performed on an existing set of to predicates to perform interpretation of tactics sketches. Results are already promising and will be further improved with the creation of dedicated domain model.

**References**


Lucchesi, M. 2002. Coaching the 3-4-1-2 and 4-2-3-1. Reedswain Publishing.


