Layered Disclosure: Revealing Agents' Internals

Patrick Riley* Peter Stone† Manuela Veloso*

* Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213-3891
{pfr, veloso}@cs.cmu.edu
† AT&T Labs — Research
180 Park Ave., room A273
Florham Park, NJ 07932
pstone@research.att.com

Abstract. A perennial challenge in creating and using complex autonomous agents is following their choices of actions as the world changes dynamically and understanding why they act as they do. This paper reports on our work to support human developers and observers to better follow and understand the actions of autonomous agents. We introduce the concept of *layered disclosure* by which autonomous agents have included in their architecture the foundations necessary to allow them to disclose upon request the specific reasons for their actions. Layered disclosure hence goes beyond standard plain code debugging tools. In its essence it also gives the agent designer the ability to define an appropriate information hierarchy, which can include agent-specific constructs such as internal state that persists over time. The user may request this information at any of the specified levels of detail, and either retroactively or while the agent is acting. We present layered disclosure as we created and implemented it in the simulated robotic soccer domain. We contribute the detailed design to support the application of layered disclosure to other agent domains. Layered disclosure played an important role in our successful development of the undefeated RoboCup champion CMUnited-99 multiagent team.

1 Introduction

A perennial challenge in creating and using complex autonomous agents is following their choices of actions as the world changes dynamically, and understanding why they act as they do. Our work focuses on environments in which agents have complex, possibly noisy, sensors and actuators. In such scenarios, even the human who develops an agent is often unable to identify what exactly caused the agent to act as it did in a given situation. This paper reports on our work to support human developers and observers to better follow and understand the actions of autonomous agents.

To this end, we introduce the concept of *layered disclosure* by which autonomous agents include in their architecture the foundations necessary to allow them to disclose to a person upon request the specific reasons for their actions. Whereas standard debugging tools allow one to trace function calls with inputs and outputs automatically, layered disclosure gives the programmer the ability to define a specific information hierarchy, which can include agent-specific constructs such as internal state that persists over time. However, the information hierarchy must be created for the specific domain. Past agent explanation paradigms (e.g. [2, 3]) have dealt with symbolic agent representations. Layered disclosure has been implemented and tested in a non-symbolic, real-time, multiagent environment.

A key component of layered disclosure is that the relevant agent information is organized in *layers*. In general, there is far too much information available to display all of it at all times. The imposed hierarchy allows the user to select at which level of detail he or she would like to probe into the agent in question. A person may then request information at any level of detail, and either retroactively or while the agent is acting. We say that the person is *probing* for information, while the agent is explicitly *disclosing* the information.

When an agent does something unexpected or undesirable, it is particularly useful to be able to isolate precisely *why* it took such an action. Using layered disclosure, a developer can probe inside the agent at any level of detail to determine precisely what needs to be altered in order to attain the desired agent behavior. For example, if a software agent is left to perform some action overnight, but fails to complete its assigned task, the developer could use layered disclosure to trace the precise reasons that the agent took each of its actions, and identify which parts of the agent need to be altered.

This paper describes our implementation of layered disclosure in the simulated robotic soccer domain. It played an important role in our successful development of the 1999 RoboCup simulator-league champion CMUnited-99 team. While the prototype implementation, including the information layers, is tailored to this domain, the layered disclosure methodology can be applied to any agent, provided that the developer defines and implements the required information hierarchy.

The remainder of this paper is organized as follows. Section 2 introduces the technical details of layered disclosure. Section 3 introduces our implementation of layered disclosure in the robotic soccer domain. Section 4 provides two detailed examples illustrating the effectiveness of layered disclosure. Sections 5 and 6 contain discussion and conclusions.

2 Layered Disclosure

In developing layered disclosure, we began with the assumption that agents' actions and the actual states of the world over time are generally observable. On the other hand, several agent characteristics are generally unobservable, including:

– The agents' sensory *perceptions*
– The agents' *internal states* (current role in team, current task assignment, etc.)
– The agents' *perceived current world states*
– The agents' *reasoning processes*

The goal of layered disclosure is to make these unobservable characteristics observable, either retroactively, or in real-time as an agent is acting. Furthermore, to avoid being overwhelmed with data, the observer must be able to probe into the agent at an arbitrary level of detail or abstraction.

There are four main steps to realizing this goal.

1. The developer must organize the agent's perception-cognition-action process in different levels of detail.
2. The agent must store a log of all relevant information from its internal state, world model, and reasoning process.
3. This log must be synchronized with a recording of the observable world (or generated in real-time).
4. An interface is needed to allow the developer to probe a given agent's internal reasoning at any time and any level of detail.

2.1 Layered Organization

The first step to implementing layered disclosure is the generation of a layered organizational structure of information to be stored for a given domain. To be the most useful, the hierrachy should contain information beyond the agent's reasoning process such as its perceptions, its internal state, and anything else that might influence its decision-making.

In particular, not all relevant information is necessarily present in an agent's action trace. For example, if an agent executes action with precondition $x > 45$ at time t, then one can conclude that at time t the variable x had a value greater than 45. However, one might want to know exactly what the value of x was at time t and how it got to be so. $x = 46$ may be a qualitatively different symptom from $x = 10,000$. Layered disclosure allows one to probe into an agent to determine the value of x at time t, and, if probing more deeply, the agent's exact sensory perceptions and/or past states which caused x to be set to its actual value at time t.

How the information hierarchy is constructed depends strongly on the architecture of the agent. For example, in our robotic soccer implementation, we use the following, where layers with greater numbers represent information that is stored at a deeper layer (scale 1–50):

Levels 1–10: The agent's abstract high-level goals.
Levels 11–20: The agent's action decisions, perhaps organized hierarchically.
Levels 21–30: Lower-level details regarding the specifics of these actions including all variable bindings.
Levels 31–40: The agent's internal state.
Levels 41–50: The agent's sensory perceptions.

If the agents uses a system which already includes a hierarchy, such as the operator hierarchy in hierarchical task network (HTN) planning [7], then this forms a nice basis on which to organize the layers. However, as argued above, just seeing which operators were executed does not always provide enough information. Therefore, more would need to be added to the information layers in order to make disclosure as useful as possible.

As another example, in a rule-based system, one would probably want to record which rules fired, what the exact values of the relevant variables were, what other rules also met their preconditions, and how any such ties were broken. In some cases, one may also want to include information on why certain rules did *not* meet their preconditions, though recording information about perceptions and internal state may allow the developer to reconstruct that information.

2.2 The Log File

Each agent's log file is its repository for all useful information that one might want to examine, either as the agent is acting or after the fact. It is generated and stored locally by the agent, or sent directly to the interface for immediate display. Each piece of information is time-stamped and tagged with its corresponding layer indicating its depth within the agents' reasoning process. The interface program can then display this information appropriately.

In general, the log file is of the format

```
<time> (<level ind>) <Text>
```

where

<time>	is the agent's conception of the current time.
<level ind>	is the information's layer.
<Text>	is arbitrary agent information to be displayed.

Note that in a distributed or asynchronous environment, the agent does not in general have access to the "real world" or global time.

2.3 Synchronization

The agent's log files, which record what would otherwise be unobservable, must be synchronized with a recording of the *observable* world. That is, the human observer needs to be able to identify what point of the log file corresponds to a given point in the recording of the world. The more exact a correspondence can be determined, the more useful the disclosure will be.

In some domains, synchronization may be difficult to achieve. Without synchronization, layered disclosure can still be used retrospectively to understand what an agent was doing given its perception of the world. However, to understand the entire loop, that is to understand whether an agent's action was appropriate to what was really going on, and to access the agent's state from a recording of the world, synchronization is needed.

In order to synchronize, the recording of the real world must contain cues or time-stamps which match up with cues or time-stamps in the agents' log files. In environments such that agents have accurate knowledge of the global system or real-world time, the synchronization can be based on this time. Otherwise, the agent must actively transmit its internal time in such a way that it is visible to the recorder in conjunction with the agent's actions. For example, in a robotics domain, one could attach lights to the robots which flash in a given sequence in order to create a visual cue to line up the agent's log file and recording of the real world. It is then the job of the interface to display to the user both the recording and the requested layered disclosure information.

2.4 Interface

The interface includes a representation of the agents' actions in the form of a recording. In addition, the interface includes a method for the user to specify an agent (if there are multiple agents) and an information layer. The interface then displays the specified layer of information for the specified agent at the moment in question. The interface can also visually separate the layers to aid the human observer.

In general, there is far too much information available to display all of it at all times. The imposed hierarchy allows the user to select at which level of detail he or she would like to probe into the agent in question.

3 Layered Disclosure in Robotic Soccer

Our prototype layered disclosure implementation is in the simulated robotic soccer domain, in particular, in the RoboCup soccer server [1, 4]. In this domain, there are 22 agents, each acting 10 times every second. Each agent gets local, incomplete perceptory information, making it impossible to determine an agent's impression of the world state based only upon the actual world state. Furthermore, it is possible for the agents to store internal state over time that can also affect their action decisions.

As laid out in Section 2 the actual world state and an agent's actions are observable (via the soccer monitor program which shows an overhead view of the soccer field), but the agents' sensory perceptions, internal states, perceived current world states, and reasoning processes are in general not observable. This lack of observability often makes it impossible to determine *why* agents acted as they did.

For example, whenever a robotic soccer agent kicks the ball towards its own goal, it is common to hear the agent's developer wondering aloud whether the agent was mistaken about its own location in the world, whether it was mistaken about the ball's or other agents' locations, or if it "meant" to kick the ball where it did, and why. Due to the dynamic, uncertain nature of the environment, it is usually impossible to reconstruct the situation exactly in order to retroactively figure out what happened. Indeed, our development of layered disclosure was inspired in part by our own inability to trace the reasons behind the actions of our own CMUnited-98 simulated agents [6].

Our layered disclosure implementation is publicly available.[1] It can and has been easily adapted for use with other RoboCup simulator teams. It works as follows.

During the course of a game, our agents store detailed records of selected information in their perceived world states, their determination of their short-term goals, and their selections of which actions will achieve these goals, along with any relevant intermediate decisions that lead to their action selections.

For example, the log file corresponding to the interface output shown in Figure 3 for agent 5 at time 838 looks like:

```
838 (35) Invalidating ball vel:0.36>0.36, vel was (1.73, -0.70)
838 (35) PB vel. est.: gpos(-32.1 -23.4),prev_seen(-33.5 -23.1)
838 (45) Sight 838.0:  B_ team:___________opp:________9__
838 (5) Mode: AM_With_Ball
838 (12) Handling the ball
838 (20) OurBreakaway() == 0
838 (25) CanDribbleTo (-22.05, -20.52): TRUE No players in cone
838 (17) handle_ball: dribbling to goal (2) -- have room
```

The content of these lines is chosen so as to be enlightening to the developer.

After the game is over, it can be replayed using the standard "logplayer" program which comes with the soccer server. Our layered disclosure module, implemented as an

[1] http://www.cs.cmu.edu/afs/cs/usr/pstone/mosaic/RoboCup/CMUnited99-sim.html

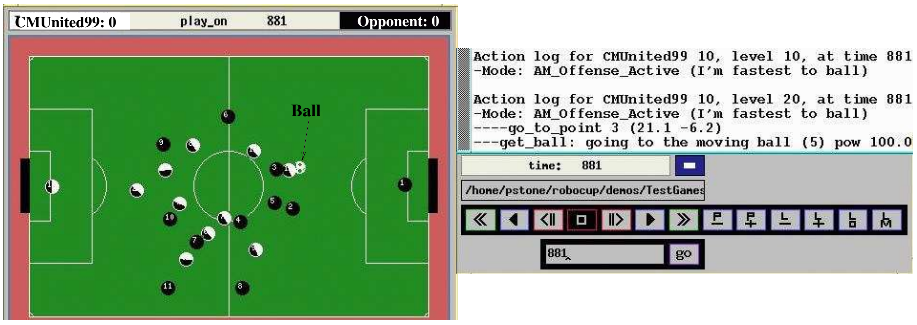

Fig. 1: The layered disclosure tool. The terminal window (top right) shows layered information for the instant shown in the graphical game display (left). On the control bar (bottom right), the "P" buttons control which player is being examined and the "L" buttons control the level of displayed information.

extension to this logplayer, makes it possible to stop the game at any point and inspect the details of an individual player's decision-making process. In this domain, the agents have an internal notion of the global time. Although it is not guaranteed to be accurate, it is sufficient for layered disclosure synchronization. Figure 1 shows our robotic soccer layered disclosure interface. For each time step, the interface strips off the times, but displays the layer and the text. The layer can be represented graphically as a series of dashes ("---") corresponding to the layer's depth.

The log file of the player being examined in Figure 1 contains the following lines. Notice that in Figure 1, only the lines at level 20 and below are displayed.

```
881 (35) My Pos: (15.85, -3.73)   angle: -24.00
881 (35) Ball Pos: (17.99, -4.87)conf: 0.95
881 (45) Sight at 880: Bv team:_  opp: 1
881 (5) Mode: AM_Offense_Active (I'm fastest to ball)
881 (15) get_ball: going to the moving ball (5) pow 100.0
881 (25) go_to_point: dash 100.0
881 (20) go_to_point 3 (21.1 -6.2)
881 (30) dashing 100.0
881 (45) Heard message of type 1 from 11
```

4 Layered Disclosure Examples

In this section we provide two examples illustrating the usefulness of layered disclosure. Both examples are drawn from the simulated robotic soccer domain.

4.1 Discovering Agent Beliefs

When observing an agent team performing, it is tempting, especially for a person familiar with the agents' architectures, to infer high level beliefs and intentions from the observed actions. Sometimes, this can be helpful to describe the events in the world, but misinterpretation is a significant danger.

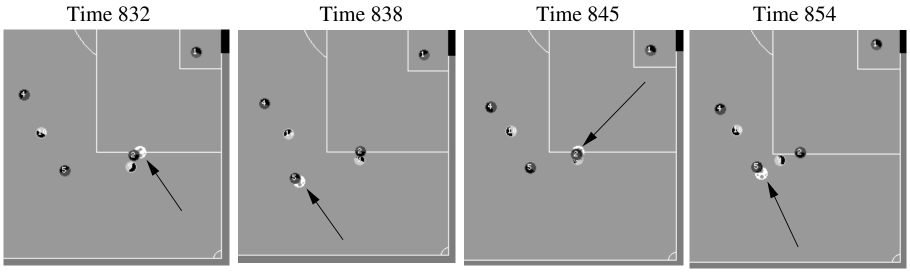

Fig. 2: Undesired passing behavior. The arrows point to the ball.

```
Action log for CMUnited99 2, level 30, at time 831
-Mode: AM_With_Ball
--Handling the ball
----OurBreakaway() == 0
---handle_ball:  need to clear
---clear_ball:  target ang == -93.0
-----starting kick to angle -93.0,
     translated to point (-6.4, -34.0)
```

```
Action log for CMUnited99 2, level 20, at time 845
-Mode: AM_With_Ball
--Handling the ball
---handle_ball:  need to clear
---clear_ball:  target ang == 24.0
----starting kick to angle 24.0,
    translated to point (-16.5, -34.0)
----kick_ball: starting kick to angle 24.0
```

```
Action log for CMUnited99 5, level 50, at time 838
-------  Invalidating ball vel
        0.36 > 0.36 thought vel was (1.73, -0.70)
-------Position based velocity estimating:
        gpos (-32.1 -23.4), prev_seen_pos (-33.5 -
23.1)
---------Sight 838.0:
            B_ team:___________opp:________9__
-Mode: AM_With_Ball
--Handling the ball
----OurBreakaway() == 0
-----CanDribbleTo (-22.05, -20.52):
             TRUE No players in cone
---handle_ball:  dribbling to goal 2
```

```
Action log for CMUnited99 5, level 20, at time 850
-Mode: AM_With_Ball
--Handling the ball
---handle_ball:  dribbling to goal 2
```

Fig. 3: Layered disclosure for the passing example (the boxes have been added for emphasis).

Consider the example in Figure 2. Here, two defenders (the darker colored players) seem to pass the ball back and forth while quite close to their own goal. In general, this sort of passing back and forth in a short time span is undesirable, and it is exceptionally dangerous near the agents' own goal. Using the layered disclosure tool, we get the information displayed in Figure 3. Note that each dash '−' represents 5 levels.

First, we see that in both cases that player number 2 was in control of the ball, it was trying to clear it (just kick it away from the goal), not pass to player number 5. Given the proximity of the goal and opponents, clearing is a reasonable behavior here. If a teammate happens to intercept a clear, then our team is still in control of the ball. Therefore, we conclude that this agent's behavior matches what we want and expect.

Next, we can see that player number 5 was trying to dribble towards the opponent's goal in both cases that he controlled the ball. There are no opponents immediately around him, and the path on the way to the goal is clear. This agent's intention is certainly reasonable.

However, player number 5 does not perform as it intended. Rather than dribbling forward with the ball, it kicked the ball backwards. This points to some problem with the dribbling behavior. As we go down in the layers, we see that the agent invalidated the ball's velocity. This means that it thought the ball's observed position was so far

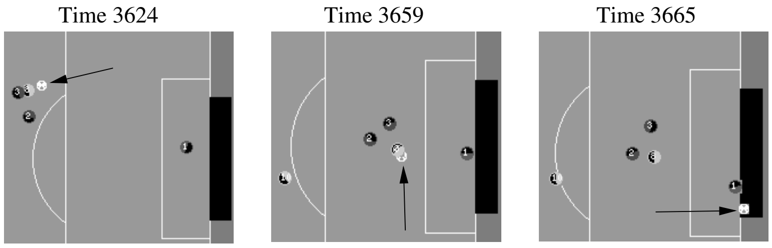

Fig. 4: Poorly performing defenders. The arrows point to the ball.

off of its predicted position that the agent's estimate for the ball's velocity could not possibly be right. The agent then computed a new estimate for the ball's velocity based on its past and current positions.

Given this estimation of the ball's velocity (which is crucial for accurate ball handling), we are led to look further into how this velocity is estimated. Also, we can compare the estimate of the velocity to the recorded world state. In the end, we find that the ball collided with the player. Therefore, it was invalid to estimate the ball's velocity based on position. In fact, this led us to more careful application of this velocity estimation technique.

Notice how useful the layering of information was here. One can first make a high level pass to understand the intentions of the agents, then examine whether thos intentions were fulfilled. If not, one can probe more deeply to discover what caused the agent to not fulfill its intention.

In this case, inferring the intentions of the players was extremely challenging given their behaviors. Without layered disclosure, the natural place to look to correct this undesirable behavior would have been in the passing decisions of the players. It would have been difficult or impossible to determine that the problem was with the estimation of the ball's velocity.

4.2 The Usefulness of Layers

The fact that the agents' recordings are layered is quite important. One important effect is that the layers allow the observer to look at just higher levels, then explore each case more deeply as required.

Consider the example depicted in Figure 4. Here, two defenders (the darker colored players) are unable to catch up and stop one offensive player with the ball, even though the defenders were in a good position to begin with.

Since this is a scenario that unfolds over many time steps, we need to be able to understand what happens over that time sequence. The first pass at this is to just look at the highest level decision. The first decision our agents make is in which "action mode" they are [6]. This decision is based on which team is controlling the ball, current location, role in the team structure, etc. Usually, the player fastest to the ball will be in "Offense_Active" mode, meaning they will try to get to the ball as quickly as possible. In a defensive situation, the second fastest player will be in "Defense_Active" mode,

Time	Player 2	Player 3
3624	-Defense_Active(poss=?)	***-Offense_Active***
3625	-Defense_Active(poss=l)	***-Offense_Active***
3626	-Defense_Active(poss=?)	***-Offense_Active***
3627	-Defense_Active(poss=l)	***-Offense_Active***
3628	***-Offense_Active***	-Defense_Active(poss=?)
3629	***-Offense_Active***	***-Offense_Active***
3630	***-Offense_Active***	-Defense_Active(poss=l)
3631	***-Offense_Active***	-Defense_Active(poss=l)
3632	***-Offense_Active***	-Defense_Active(poss=l)
3633	***-Offense_Active***	-Defense_Active(poss=l)
3634	***-Offense_Active***	-Defense_Active(poss=l)
3635	***-Offense_Active***	-Defense_Active(poss=l)
3636	-Defense_Active(poss=l)	***-Offense_Active***
3637	-Defense_Active(poss=l)	***-Offense_Active***
3638	-Defense_Active(poss=l)	***-Offense_Active***
3639	***-Offense_Active***	-Defense_Active(poss=l)
3640	***-Offense_Active***	-Defense_Active(poss=?)
3641	-Defense_Active(poss=l)	-Defense_Active(poss=l)
3642	-Defense_Active(poss=?)	-Defense_Active(poss=l)
3643	***-Offense_Active***	***-Offense_Active***
3644	***-Offense_Active***	-Defense_Active(poss=l)
3645	***-Offense_Active***	-Defense_Active(poss=l)
3646	-Defense_Active(poss=l)	-Defense_Active(poss=l)
3647	***-Offense_Active***	***-Offense_Active***
3648	-Defense_Active(poss=?)	***-Offense_Active***
3649	***-Offense_Active***	***-Offense_Active***
3650	-Defense_Active(poss=l)	***-Offense_Active***
3651	***-Offense_Active***	-Defense_Active(poss=l)
3652	***-Offense_Active***	-Defense_Active(poss=l)
3653	***-Offense_Active***	-Defense_Active(poss=?)
3654	***-Offense_Active***	-Defense_Active(poss=?)
3655	***-Offense_Active***	-Defense_Active(poss=?)
3656	***-Offense_Active***	-Defense_Active(poss=?)
3657	***-Offense_Active***	***-Offense_Active***
3658	***-Offense_Active***	***-Offense_Active***
3659	***-Offense_Active***	***-Offense_Active***

Fig. 5: Layered disclosure for the defending example (the bold italics have been added for emphasis).

which means basically to get in the way of the player with the ball, without actively trying to steal it.

The output of just the highest level from the layered disclosure tool is depicted in Figure 5. There are two things to notice. First, the agents change roles many times over this sequence. Secondly, the agents' are often unsure about which side is in control of the ball (the 'poss=' field).

This constant changing of mode causes a problem for the agents. "Offense_Active" and "Defense_Active" modes lead the players to move to different spots on the field. By switching back and forth, the agents will waste a great deal of time turning to face the direction they want to go instead of actually going. Therefore, the agents do not catch up.

Noticing that the 'poss=' field is also in flux allows further diagnosis of the problem. The decision about what mode to go into is sometimes affected by which team the agent believes is controlling the ball. Realizing that this value is often unknown should lead to changes in the way that value is determined, or changes in the manner in which it is used.

In this case, making use of layered disclosure to examine just the high-level reasoning decisions of a pair of agents allows us to focus on a problem that would have otherwise been easily overlooked.

5 Discussion

Most existing debugging tools aim to automatically provide details of a program's execution. One common method is to store the entire program call stack with the arguments to each function call. In agent-based applications, it is also possible to automatically store either (i) the agent's entire reasoning process that leads directly to its actions, or (ii) just the agent's perceptions and internal state at every action opportunity.

For example, if behaviors are generated by planning operators such as in an HTN planner, (i) could be accomplished by storing the preconditions and effects of all selected operators. However, as argued in Section 2.1, the preconditions of selected actions do not necessarily disclose all relevant information pertaining to the agent.

Option (ii) allows one to recreate the agent's reasoning process and resulting actions by stepping through the program after the fact. While this option might be appropriate for agents with small states, in domains such as robotic soccer in which a large amount of information is relevant to each decision, our approach is more efficient both in terms of storage space and amount of information that needs to be conveyed to a human being; the developer can identify precisely what information is likely to be useful. In addition, storing only the internal state for subsequent reconstruction of the agent's behaviors does not allow for the agent also being able to disclose its internal state in real time as the action is proceeding.

As opposed to these automatic debugging options, layered disclosure requires a step on the part of the developer, namely the explicit definition and inclusion of an information hierarchy within the agent. In return, the developer gets full control over what information is disclosed by the agent. Our use of layered disclosure in the robotic soccer domain represents an efficient middle ground between storing the agents' entire internal states and storing just their actions.

Layered disclosure is potentially applicable in any agent-based domain. While our application is in a multiagent environment, it is inherently implemented on an individual agent: each agent discloses its own internal state. In order to take advantage of layered disclosure, one only needs to provide support for agents to store and disclose their perceptions, internal states, perceived current world states, and reasoning processes; as well as a method for synchronization between the agent's logfile and a recording of the world.

6 Conclusion and Future Work

Layered disclosure has proven to be very useful to us in our development of simulated robotic soccer agents. In particular, we used it as an integral part of our development of the 1999 RoboCup champion CMUnited-99 simulated robotic soccer team [5].

CMUnited-99 was built upon the CMUnited-98 team which won the RoboCup-98 simulator competition [6]. CMUnited-98 became publicly available after the 1998 competition so that other people could build upon our research. Thus, we expected there to be several teams at RoboCup-99 that could beat CMUnited-98, and indeed there were.

With this expectation, we knew that CMUnited-99 needed to improve upon CMUnited-98. However, even when we observed undesirable behaviors in CMUnited-98, it was not always clear what was going wrong. Using layered disclosure, we were able to identify and implement several significant improvements. CMUnited-99 was able to consistently beat CMUnited-98 by a significant margin and it went on to win the RoboCup-99 championship, outscoring its opponents by a combined score of 110–0.

While the prototype layered disclosure implementation is designed for a multi-agent system, it only allows the user to probe one agent at a time. A useful extension would be to allow the user to select any subset of agents and probe them all simultaneously. Coordination procedures could then be more easily understood.

Another interesting extension would be to allow the user more power in exploring the hierarchy. Currently, the user can select a layer of information to display. However, there may be only a small fraction of the information at that layer which is relevant for the current situation. Making the information hierarchy into a tree or a DAG could allow the user to further limit the amount of information displayed. However, one must be careful to keep it easy and efficient for the user to access all of the information.

Layered disclosure also has the potential to be used as a vechicle for interactive agent control. When coupled with an interface for influencing agent behaviors, layered disclosure could be used to allow the interleaving of autonomous and manual control of agents. Giving the user the power of reasonable abstractions allows them to effectively intercede in the agents' behaviors. For example, given a set of robots autonomously cleaning the floors in a large building, a person might want to monitor agents' perceptions of which floors are in most urgent need of attention. Upon noticing that one of the agents is ignoring an area that is particularly dirty, the person could determine the cause of the agent's oversight and manually alter the internal state or goal stack within the agent in real time.

We envision that layered disclosure will continue to be useful in agent development projects, particularly with complex agents acting in complex, dynamic environments. We also plan to begin using layered disclosure in interactive semi-autonomous agent-control scenarios.

References

1. Emiel Corten, Klaus Dorer, Fredrik Heintz, Kostas Kostiadis, Johan Kummeneje, Helmut Myritz, Itsuki Noda, Patrick Riley, Peter Stone, and Travlex Yeap. Soccer server manual, version 5.0. Technical Report RoboCup-1999-001, RoboCup, 1999. At URL http://ci.etl.go.jp/~noda/soccer/server/Documents.html.
2. Adele E. Howe, Robert St. Amant, and Paul R. Cohen. Integrating statistical methods for characterizing causal influences on planner behavior over time. In *Proceedings of the Sixth IEEE International Conference on Tools with AI*, 1994.
3. W. Lewis Johnson. Agents that learn to explain themselves. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1257–1263, 1994.

4. Itsuki Noda, Hitoshi Matsubara, Kazuo Hiraki, and Ian Frank. Soccer server: A tool for research on multiagent systems. *Applied Artificial Intelligence*, 12:233–250, 1998.
5. Peter Stone, Patrick Riley, and Manuela Veloso. The CMUnited-99 champion simulator team. In Manuela Veloso, Enrico Pagello, and Hiroaki Kitano, editors, *RoboCup-99: Robot Soccer World Cup III*, Berlin, 2000. Springer Verlag.
6. Peter Stone, Manuela Veloso, and Patrick Riley. The CMUnited-98 champion simulator team. In Minoru Asada and Hiroaki Kitano, editors, *RoboCup-98: Robot Soccer World Cup II*. Springer Verlag, Berlin, 1999.
7. David E. Wilkins. Domain-independent planning: Representation and plan generation. *Artificial Intelligence*, 22:269–301, 1984.