Cyberoos'99: Tactical Agents in the RoboCup Simulation League

Mikhail Prokopenko, Marc Butler, Wai Yat Wong, Thomas Howard

Applied Artificial Intelligence Project
CSIRO Mathematical and Information Sciences
Locked Bag 17, North Ryde, NSW 1670, Australia

Abstract. This paper describes a framework for formalising tactical reasoning in dynamic multi-agent systems, populated by synthetic (software) agents. The proposed framework is based on a hierarchy of synthetic agent architectures and is expressive enough to capture a subset of desirable properties from both the situated automata and subsumption-style architectures, while retaining the rigour and clarity of logic-based possible worlds semantics. This framework is successfully realised in the RoboCup Simulation League domain. Not only did it provide a solid design approach to object-orientation, but it also enabled incremental implementation and testing of software agents and their modules. In particular, the framework allowed us to correlate enhancements in the agent architecture with tangible improvements in team performance. Cyberoos98 was 3rd place winner of the Pacific Rim series at PRICAI-98. Cyberoos99 finished in the top 18 of the RoboCup-99.

1 Introduction

The principal aim of this paper is to illustrate how declarative agent specifications may facilitate design and implementation of software agents capable of tactical reasoning. Some of the architectures are well-known - for example, variants of tropistic and hysteretic agents, enabling typical perception-action feedback loop and behavioural subsumption are discussed in [2, 5, 7]. We attempted to extend these results by including new agent types (task-oriented and process-oriented agents), facilitating reasoning about tactical activities in context of multi-agent teamwork. These agent architectures allow us to capture a number of desirable properties (reactive plans, ramifications, task-oriented and, potentially, goal-directed behaviour) by embedding them in situated behaviours.

2 Situated Agent Architecture

2.1 Tropistic Agent

Following [5], we formally describe a *Tropistic* agent as a tuple $\boldsymbol{A}_T$

$$\langle \boldsymbol{C}, \boldsymbol{S}, \boldsymbol{E}, \mathit{sense}, \mathit{tropistic\text{-}behaviour}, \mathit{response} \rangle,$$

where $\boldsymbol{S}$ is a set of agent sensory states, $\boldsymbol{E}$ is a set of effectors, and $\boldsymbol{C}$ is a communication channel type. A sensory function is defined as $\mathit{sense}: \boldsymbol{C} \rightarrow \boldsymbol{S}$. Activity of a *Tropistic* agent is characterised by $\mathit{tropistic\text{-}behaviour}: \boldsymbol{S} \rightarrow \boldsymbol{E}$. By allowing the set $\boldsymbol{E}$ to include composite effectors $\boldsymbol{e1}; \boldsymbol{e2}$, where $\boldsymbol{e1} \in \boldsymbol{E}$, $\boldsymbol{e2} \in \boldsymbol{E}$, we can implicitly account for the case of reactive planning - when a situated agent reacts to stimuli $\boldsymbol{S}$ with an n-length sequence of

effectors. The *Tropistic* agent executes its behaviour as a sequence of commands encoded by *response*: $\boldsymbol{E} \rightarrow \boldsymbol{C}$ and sent to the simulator.

The *Tropistic* agent must manage two concurrent and asynchronous activities - one corresponding to receiving sensory information from the Soccer Server, and the other related to sending appropriate atomic commands to the Server. We chose to implement the required parallelism via threads, as they appeared to be intuitively appropriate for the requirements, and (assuming implementation in C or C++) have native support in the Solaris operating system. Threads allowed us to use non-blocking I/O, eliminating the necessity of timing the sensory thread. In addition, we were not limited by the number of available signals, provided by the operating system. The nature of tropistic behaviour and asynchronous character of client-server communication make precise timing imperative but problematic. Some of the observed difficulties arise due to 1) receiving outdated sensory information and 2) sending too many commands in a given simulation cycle. Non-blocking I/O network access complemented by an appropriate technique for processing of pending messages addresses the first challenge, rescuing the agents from the "living-in-the-past" syndrome. The second difficulty can be rectified if the acting thread independently schedules agent responses.

The most important examples of tropistic behaviour exhibited by a Cyberoos99 agent are obstacle avoidance, ball chasing, and (a goalkeeper) catch.

2.2 Hysteretic Agent

A *Hysteretic* agent is defined here as a reactive agent maintaining internal state $\boldsymbol{I}$ and using it as well as sensory states $\boldsymbol{S}$ in activating effectors $\boldsymbol{E}$; i.e. its activity is characterised by *hysteretic-behaviour*: $\boldsymbol{I}$ x $\boldsymbol{S} \rightarrow \boldsymbol{E}$. A memory update function maps an internal state and an observation into the next internal state, i.e. it defines *update*: $\boldsymbol{I}$ x $\boldsymbol{S} \rightarrow \boldsymbol{I}$. A *Hysteretic* agent reacts to stimuli $\boldsymbol{s}$ sensed by *sense*($\boldsymbol{c}$) and activates effectors $\boldsymbol{e}$ according to *hysteretic-behaviour*($\boldsymbol{i}$, $\boldsymbol{s}$). This class extends its superclasses by adding *hysteretic-behaviour* and *update* functions, while retaining all previously defined functions (i.e., it is a sub-class of the *Tropistic* agent). So the *Hysteretic* agent is defined as a tuple $\boldsymbol{A}_{\boldsymbol{H}}$

$$\langle \boldsymbol{C}, \boldsymbol{S}, \boldsymbol{E}, \boldsymbol{I}, \textit{ sense}, \textit{tropistic-behaviour}, \boldsymbol{\textit{hysteretic-behaviour}}, \boldsymbol{\textit{update}}, \textit{response} \rangle$$

where the ***bold*** style indicates newly introduced functions.

The *hysteretic-behaviour* is implemented as a (temporal) production system (TPS). Whenever the TPS fires a rule (a behaviour instantiation, expressed in terms of partial sensory and internal states, ie., in the form similar to situated automata condition-action pairs [3]), a sequence of atomic commands is placed into a queue for subsequent and timely execution. Implementation of the TPS enables monitoring of currently progressing actions, thus providing an explicit account of temporal continuity for actions with duration [7, 8], and allows us to embed actions ramifications and interactions [4, 6, 7]. For example, a dribbling action will not be invoked during shooting or passing.

It is worth pointing out that the architecture $\boldsymbol{A}_{\boldsymbol{H}}$ can be viewed as a subsumption architecture [1] as well. It allows us to easily express desired subsumption dependencies between the *Hysteretic* and *Tropistic* levels. More precisely, resolution of a possible conflict between behaviour instantiations $\boldsymbol{e1}$ = *tropistic-behaviour*($\boldsymbol{s}$) and $\boldsymbol{e2}$ = *hysteretic-behaviour*($\boldsymbol{i}$, $\boldsymbol{s}$) triggered by the same sensory input $\boldsymbol{s}$, is dependent on the internal state $\boldsymbol{i}$ - leading to inhibition of the simpler level behaviour, if necessary. For instance, tropistic

chase is suppressed if a team-mate has possession of the ball. A Cyberoos99 agent displays quite a few interesting instantiations of hysteretic behaviour, eg., dribble around opponents toward an enemy goal, intercepting a fast moving ball, controlling and turning with the ball, resultant-vector passing, shooting at goal along a non-blocked path, etc.

3 Tactical Agent Architecture

3.1 Task-Oriented Agent

The behaviour functions of the situated agents, described in the previous section, are uniformly defined across their respective domains and ranges. This means that the set of all behaviour instantiations $\boldsymbol{H} = \{(\boldsymbol{i}, \boldsymbol{s}, \boldsymbol{e})$: $\boldsymbol{e} = \textit{hysteretic-behaviour}(\boldsymbol{i}, \boldsymbol{s})\}$ is not partitioned or structured otherwise. In other words, all agent's behaviour instantiations (action rules) are always enabled. Sometimes, however, it is desirable to disable all but a subset of behaviour instantiations - for example, when a tactical task requires concentration on a specific activity. The following agent class - *Task-Oriented* agent - is intended to capture this feature, while retaining properties of the *Hysteretic* agent (i.e., it is a sub-class of the latter). We define the architecture of a *Task-Oriented* agent as the tuple $\boldsymbol{A}_{TO}$

<$\boldsymbol{C}$, $\boldsymbol{S}$, $\boldsymbol{E}$, $\boldsymbol{I}$, $\boldsymbol{T}$, *sense*, *tropistic-behaviour*, *hysteretic-behaviour*, *update*, ***decision***, ***combination***, *response*>

A *Task-Oriented* agent incorporates a set of task states $\boldsymbol{T}$. It uses the functions *decision*: $\boldsymbol{I}$ x $\boldsymbol{S}$ x $\boldsymbol{T} \rightarrow \boldsymbol{T}$ and *combination*: $\boldsymbol{T} \rightarrow \boldsymbol{2}^{\boldsymbol{H}}$ to decide which subset of behaviour instantiations (a task) is appropriate at a particular internal state, given sensory inputs.

Implementation of task-orientation requires some adjustments to the TPS. The TPS traces action rules whose actions may be in progress, and checks, in addition, whether a rule is valid with respect to a current task. The hysteretic behaviour instantiations mentioned in the previous section (dribbling, intercepting, etc.) are *combined* in corresponding tasks and can be selected by a Cyberoos99 agent in real-time. For example, sharp-dribbling is implemented as a task, enabling relevant (hysteretic) twists and faint moves.

3.2 Process-Oriented Agent

The *Task-Oriented* agent is capable of performing certain tactical combinations ("set pieces") in real-time by activating only a subset of its behaviour functions, and thus concentrating only on a specified task. Upon making a new decision, the agent switches to another task. In general, there is no dependency or continuity between consecutive tasks. This is quite suitable in complex and/or unexpected situations requiring a swift reaction. However, in some cases it is desirable to exhibit a more coherent agent behaviour.

A new notion, a process, is intended to capture this kind of coherent behaviour - when an agent is engaged in an activity requiring several tasks. A process constrains a set of possible tasks without specifying an exact sequential or tree-like ordering - an appropriate tactical scheme comprising a few tactical elements may simply suggest for an agent a possible subset of decisions, leaving some of them optional. For example, a penetration through centre of an opponent penalty area may require from agent(s) to employ a certain tactics - a certain set of elementary tactical tasks (dummy-runs, wall-passes, short-range dribbling) - and disregard for a while another set of tasks. It is worth noting that whereas a team's tactical formation is typically a static view of responsibilities and relationships, process-orientation is a dynamic view of how this formation delivers tactical solutions.

The architecture of a *Process-Oriented* agent can be defined as the tuple $\boldsymbol{A_{PO}}$

$$\langle \boldsymbol{C}, \boldsymbol{S}, \boldsymbol{E}, \boldsymbol{I}, \boldsymbol{T}, \boldsymbol{P}, \textit{sense}, \textit{tropistic-behaviour}, \textit{hysteretic-behaviour}, \textit{update}, \textit{decision}, \textit{combination}, \boldsymbol{engage}, \boldsymbol{tactics}, \textit{response} \rangle$$

A *Process-Oriented* agent maintains a process state $\boldsymbol{P}$. It uses the functions *engage*: $\boldsymbol{I} \times \boldsymbol{S} \times \boldsymbol{T} \times \boldsymbol{P} \rightarrow \boldsymbol{P}$ and *tactics*: $\boldsymbol{P} \rightarrow \boldsymbol{2^T}$ to select a subset of tasks, given current internal, sensory, task and process states. Several tactical processes can be selected by Cyberoos99 agents in real-time: *Advance, Dispatch,* and *Penetration*. Importantly, tasks encapsulated in a process are not temporally ordered - instead, they make up a (currently) relevant *tactical* selection, arranged in appropriate decision-making order - for instance, an agent *engaged* in the *Dispatch* process considers the passing task before dribbling tasks, while the *Penetration* process prefers dribbling to crossing and disregards passing at all.

4 Conclusions

The notions of task and process specified for each agent open a way to formally introduce a group of agents sharing the same task or the same process, and enable formal tactical reasoning about multi-agent teamwork. It is important to realise that teamwork is formalised on the system level. In other words, the overall team behaviour/tactics emerges only as a result of agent interactions.

The described hierarchical framework provided a solid design approach to object-orientation, and enabled rigorous incremental implementation and testing of software agents and their modules. We used C++ as the implementation language; the development environment was Solaris 2.5 and GNU g++ 2.8.2 on SPARC workstations. The team took 3rd place at the PRICAI-98 Pacific Rim series. Cyberoos99 qualified for the RoboCup-99 finals, where the team played 10 games (both qualification and consolation), winning 5 of them with a total score 97:15.

References

1. Brooks, R.A. Intelligence Without Reason. In Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, 569-595 Morgan Kaufmann, 1991.
2. Genesereth, M.R., and Nilsson, N.J. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, 1987.
3. Kaelbling, L.P. and Rosenschein, S.J. Action and planning in embedded agents. In Maes, P. (ed) *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, pages 35-48, 1990.
4. Prokopenko, M., Jauregui, V. Reasoning about Actions in Virtual Reality. In Proceedings of the IJCAI-97 Workshop on Nonmonotonic Reasoning, Action and Change, 159-171. Nagoya 1997.
5. Prokopenko, M., Kowalczyk R., Lee M., Wong, W.-Y. Designing and Modelling Situated Agents Systematically: Cyberoos'98. In Proceedings of the PRICAI-98 Workshop on RoboCup, 75-89. Singapore 1998.
6. Prokopenko, M. Situated Reasoning in Multi-Agent Systems. In AAAI Technical Report SS-96-05, the AAAI-99 Spring Symposium on Hybrid Systems and AI, 158 - 163. Stanford 1999.
7. Prokopenko, M., Butler M. Tactical Reasoning in Synthetic Multi-Agent Systems: a Case Study. In Proceedings of the IJCAI-99 Workshop on Nonmonotonic Reasoning, Action and Change. Stockholm, 1999.
8. Sandewall, E. Logic-based Modelling of Goal-Directed Behaviour. *Linköping electronic articles in Computer and Information science*, (2):19 1997.