# Tactical Reasoning in Synthetic Multi-Agent Systems: a Case Study

**Mikhail Prokopenko, Marc Butler**

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

## Abstract

The paper describes a hierarchy of logic-based agent architectures and proposes a framework for formalising tactical reasoning in dynamic multi-agent systems populated by synthetic (software) agents. We first consider basic types of situated agent architectures and their corresponding action theories. Then more complex tactical agent architectures suitable for high-level reasoning are formally defined. This approach 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 representation. In addition, it allows us to systematically and uniformly embed reactive plans, ramifications, task-oriented and, potentially, goal-directed behaviour. The framework is successfully realised in the RoboCup Simulation League domain.

## 1 Introduction

The RoboCup Synthetic Agent Challenge 97 highlighted multi-agent architectures, enabling real-time planning and plan execution in service of teamwork, as a specific research target [Kitano et al., 1997]. Multi-agent teamwork, and interactivity in dynamic systems in general, can be formalised in different ways. Approaches developed in the framework of Reasoning about Action are mostly logic-based, rely on a centralised world model, and try to (explicitly) capture various aspects of rationality. Another methodology evolved in the field of Distributed Multi-agent Systems. It usually considers autonomous agents reacting to changes in external environment and (ideally) exhibiting emergent behaviour.

Behavioural and multi-agent approaches to artificial intelligence often feature *situatedness* of agents reacting to changes in environment instead of reliance on abstract representation and inferential reasoning [Brooks, 1991; Kaelbling and Rosenschein, 1990; Steels, 1990]. Under certain assumptions, situated behaviours can be proven to be correct with respect to a theory of actions [Baral and Son, 1996], and in some cases can be derived from it [Prokopenko and Jauregui 1997].

The principal aim of this paper is to relate declarative agent specifications and situated behaviours in the context of multi-agent teamwork, where tactical reasoning is required. In other words, we will illustrate how an underlying abstract representation can facilitate design of situated agent architectures that can support coordination of group activities. The approach can specifically assist in uniformly mapping logic theories of actions to agent architectures, where desirable properties (reactive plans, ramifications, task-oriented and goal-directed behaviour) are embedded in situated behaviours.

## 2 Situated Agent Architecture

In this section we define various types of situated agent architectures and analyse their formal properties. Some of the architectures are well-known – for example, variants of tropistic and hysteretic agents are discussed in [Genesereth and Nilsson, 1987; Prokopenko et al., 1998]. We will attempt to extend these results by including new agent types (task-oriented and process-oriented agents) enabling reasoning about tactical activities in context of a multi-agent teamwork.

### 2.1 Environment Simulator

Following [Prokopenko et al., 1998; Prokopenko, 1999], we define a *Simulator* agent as a tuple $\boldsymbol{A_S}$

$$<\boldsymbol{W}, \boldsymbol{P}, \boldsymbol{A}, \boldsymbol{E}, \boldsymbol{C}, \mathit{view}, \mathit{projection}, \mathit{send}, \mathit{receive}, \mathit{do}>,$$

where $\boldsymbol{W}$ is a set of all external states, $\boldsymbol{P}$ is a set of all possible partitions of $\boldsymbol{W}$, $\boldsymbol{A}$ is a set of situated agents, $\boldsymbol{E}$ is a set of effectors, and $\boldsymbol{C}$ is a communication channel type. Function *view* structures situated agent perceptions by selecting a partition of external states for each agent. In other words, it maps an agent into an external states partition and defines *view*: $\boldsymbol{A} \rightarrow \boldsymbol{P}$.

Dependent on a current situation in the synthetic world, the *Simulator* determines which particular element from a viewable partition is currently observable by every situated agent in $\boldsymbol{A}$. In other words, the *Simulator* projects an external state and a situated agent into an element of the viewable partition of external states, using *projection*: $\boldsymbol{W} \times \boldsymbol{A} \rightarrow \boldsymbol{2^W}$, where $\boldsymbol{2^W}$ is the power-set of $\boldsymbol{W}$. The exact range of the *projection* function is the external states partition selected by *view* from the set $\boldsymbol{P}$ of all possible partitions of $\boldsymbol{W}$. More precisely,

$$\forall\, \boldsymbol{w} \in \boldsymbol{W}, \forall\, \boldsymbol{a} \in \boldsymbol{A}, \quad \mathit{projection}(\boldsymbol{w}, \boldsymbol{a}) \in \mathit{view}(\boldsymbol{a})$$

The projected partition element is a set of external states ($\mathit{projection}(\boldsymbol{w}, \boldsymbol{a}) \subseteq \boldsymbol{W}$), and is sent by the *Simulator* to the situated agent by means of the function *send*: $\boldsymbol{A} \times \boldsymbol{2^W} \rightarrow \boldsymbol{C}$. We will presume that situated agents are able to decode $\mathit{projection}(\boldsymbol{w}, \boldsymbol{a})$ from the input message, and respond back to the *Simulator* with an effector name. The received communication is decoded by *receive*: $\boldsymbol{A} \times \boldsymbol{C} \rightarrow \boldsymbol{E}$, and the communicated effector is processed by the function *do*: $\boldsymbol{E} \times \boldsymbol{W} \rightarrow \boldsymbol{W}$, which maps each effector and an external state into the next state.

## 2.2 Tropistic Agents

Having defined the architecture of the *Simulator* agent, 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, and $\boldsymbol{C}$ and $\boldsymbol{E}$ denote the same components as before. The sensory function is defined as *sense*: $\boldsymbol{C} \rightarrow \boldsymbol{S}$, where an element of $\boldsymbol{C}$ is expected to carry the information on $\mathit{projection}(\boldsymbol{w}, \boldsymbol{a})$. Activity of the agent is characterised by *tropistic-behaviour*: $\boldsymbol{S} \rightarrow \boldsymbol{E}$. By allowing the set $\boldsymbol{E}$ to include composite effectors $\boldsymbol{e_1}; \boldsymbol{e_2}$, where $\boldsymbol{e_1} \in \boldsymbol{E}$, $\boldsymbol{e_2} \in \boldsymbol{E}$, we can implicitly account for the case of tropistic planning - when a situated agent reacts to stimuli $\boldsymbol{S}$ with an n-length sequence of effectors. The *response* function communicates the selected behaviour instantiation to the *Simulator* by encoding *response*: $\boldsymbol{E} \rightarrow \boldsymbol{C}$.

In practice, it is almost impossible to express each instantiation $(\boldsymbol{e}, \boldsymbol{s})$ of the *tropistic-behaviour* function $\boldsymbol{e} = \mathit{tropistic\text{-}behaviour}(\boldsymbol{s})$ in terms of complete sensory states. Instead, we represent such behaviour instantiations in terms of partial sensory states. For example, the following rule, given in the form similar to control rules [Baral and Son, 1996] or condition-action pairs [Kaelbling and Rosenschein, 1990], describes behaviour instantiations:

**if** [SeeBall: (distance, direction) $\wedge$ Far(distance)]
**then** *turn*(direction); *dash*(2*distance)

The bracketed component on the left-hand side correspond to elements of $\boldsymbol{S}$ and has to be evaluated as true in order to activate effector(s) on the right-hand side. A sentence $\alpha$ in this component specifies the set of states from $\boldsymbol{S}$ consistent with $\alpha$. In other words, it specifies a partial sensory state. Technically, each premise could be represented by a DNF, where each conjunct describes a complete sensory state. The DNF may be divided into a number of conjunctive premises, where each modified premise can be treated as a set of atomic formulae describing a complete sensory state.

## 2.3 Hysteretic Agents

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*: $\boldsymbol{I} \times \boldsymbol{S} \rightarrow \boldsymbol{E}$. Again, we allow the set $\boldsymbol{E}$ to include composite effectors $\boldsymbol{e_1}; \boldsymbol{e_2}$, where $\boldsymbol{e_1} \in \boldsymbol{E}$, $\boldsymbol{e_2} \in \boldsymbol{E}$, covering the case of hysteretic planning. A memory update function maps an internal state and an observation into the next internal state, i.e. it defines *update*: $\boldsymbol{I} \times \boldsymbol{S} \rightarrow \boldsymbol{I}$. A *Hysteretic* agent reacts to stimuli $\boldsymbol{s}$ sensed by $\mathit{sense}(\boldsymbol{c})$ and activates effectors $\boldsymbol{e}$ according to $\mathit{hysteretic\text{-}behaviour}(\boldsymbol{i}, \boldsymbol{s})$. The agent neither has full knowledge about the state $\mathit{do}(\boldsymbol{e}, \boldsymbol{w})$ obtained by the *Simulator*, nor reasons about the transition. The next interaction with the world may bring partial knowledge about its new state.

The *Hysteretic* agent class extends its superclasses by adding the *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 $\boldsymbol{A_H}$

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

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

*Hysteretic-behaviour* instantiations may be represented in terms of partial internal and sensory states as well. For example, the following rule describes a *hysteretic-behaviour* instantiation, where the effector on the right-hand side is composite:

**if** [Orientation: angle] **and**
[SeeBall: (b, dir) $\wedge$ Close(b)]
**then** *weak_kick*(angle - dir); *turn*(angle)

Two bracketed components on the left-hand side correspond to elements of $\boldsymbol{I}$ and $\boldsymbol{S}$ respectively.

It is worth noting that the architecture $\boldsymbol{A_H}$ can be viewed as a subsumption architecture [Brooks 1991] 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 $\boldsymbol{e_1} = \mathit{tropistic\text{-}behaviour}(\boldsymbol{s})$ and $\boldsymbol{e_2} = \mathit{hysteretic\text{-}behaviour}(\boldsymbol{i}, \boldsymbol{s})$ 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 (figure 1).

An *Extended Hysteretic* agent $\boldsymbol{A_{EH}}$ is derived from the *Hysteretic* agent. Its architecture contains two additional communication components *notify* and *listen*, and is defined as a tuple

<***C***, ***S***, ***E***, ***I***, *sense, tropistic-behaviour, hysteretic-behaviour, update,* ***notify***, ***listen***, *response*>,

where the communication functions are responsible for dealing with outcoming and incoming messages exchanged among situated agents (rather than between a *Simulator* and a situated agent). The *listen* function is specialised from the *sense* function, and *notify* function is a specialised *hysteretic-behaviour*. The reason for introducing these communication functions is that domain constraints may influence internal variables of other agents or require invocation of other agents' actions. The distinction between *structural ramifications* when "the action can affect features of other objects than those which occur as arguments of the action" and *local ramifications* involving only "features of the argument objects" was identified in [Sandewall, 1994]. For example, the following domain constraint

$$\mathbf{H}(\text{t, near(x): y}) \leftrightarrow \mathbf{H}(\text{t, near(y): x})$$

demands from a model to include the atomic formula near(B): A, whenever it contains the atomic formula near(A): B. Therefore, at the moment when agent A evaluates near(A): B as true (either by sensing a new observation, or by updating an internal variable), another agent (B in this case) needs to be notified. If the agent B has limited sensory capabilities (preventing, for example, a direct sensing of near(B): A), then the communication is the only way of ensuring a synchronous assignment.

It is worth noting that "listening" to a message is a form of sensing, and "speaking" is a form of action [Parsons, Sierra, and Jennings, 1998]. Therefore, incoming messages can be sensed (*listen*-ed) by a suitable sensor, let us say, Told: ***e***, and outcoming messages can be sent by the specialised behaviour *notify* activating a suitable effector, let us say, Tell(g, ***e***), where *e* is an effector name, and g is a name of a receiving agent. For example,

**if** [PassingTo: n ] **and**
[SeePartner: (n, distance, angle) ∧ SeeBall: (dist, dir) ∧ NearBall(n)]
**then** Tell(NameOf(n), *turn*(angle - dir)

**if** [LookingForBall] **and** [Told: *turn*(x)] **then** *turn*(x)

An agent may send a message to itself. An execution of a communicated effector modifies internal variables of the receiving agent.

# 3 Tactical Agent Architecture

## 3.1 Task-Oriented Agents

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 sets of all behaviour instantiations $\boldsymbol{B}$ = {(***s***, ***e***): ***e*** = *tropistic-behaviour*(***s***)} and $\boldsymbol{H}$ = {(***i***, ***s***, ***e***): ***e*** = *hysteretic-behaviour*(***i***, ***s***)} are not partitioned or structured otherwise. In other words, all agent's behaviour instantiations are always enabled. This may be acceptable in case of tropistic behaviour. 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 *Extended Hysteretic* agent (i.e., it is a sub-class of the latter). We define the architecture of a *Task-Oriented* agent as the tuple $\boldsymbol{A_{TO}}$

<***C***, ***S***, ***E***, ***I***, ***T***, *sense, tropistic-behaviour, hysteretic-behaviour, update,* ***decision***, ***combination***, *notify, listen, response*>

A *Task-Oriented* agent incorporates a set of task states ***T***. It uses the functions *decision*: $\boldsymbol{I} \times \boldsymbol{S} \times \boldsymbol{T} \rightarrow \boldsymbol{T}$ and *combination*: $\boldsymbol{T} \rightarrow \boldsymbol{2^H}$ to decide which subset of behaviour instantiations (a task) is appropriate at a particular internal state given sensory inputs. In other words, a task activates a subset of behaviour instantiations (rules) (figure 1).

The architecture $\boldsymbol{A_{TO}}$ can be extended to a subsumption-style architecture too. Possible subsumption dependencies between the *Task-Oriented* level on one hand and more simple levels on another are easily expressed if another function, *task-oriented-behaviour* $\boldsymbol{I} \times \boldsymbol{S} \times \boldsymbol{T} \rightarrow \boldsymbol{E}$, is introduced explicitly. For example, a possible conflict between behaviour instantiations $\boldsymbol{e_1}$ = *tropistic-behaviour*(***s***), $\boldsymbol{e_2}$ = *hysteretic-behaviour*(***i***, ***s***), and $\boldsymbol{e_3}$ = *task-oriented-behaviour*(***i***, ***s***, ***t***), triggered by the same sensory input ***s*** and the internal state ***i***, can be resolved according to the task state ***t***, leading to inhibition of the simpler level(s) behaviour when necessary.

The *Task-Oriented* agent cannot share the decision-making process with other situated agents. Although it is possible for the agent to *notify* other agents as to what effectors to activate, and to *listen* to such notifications, the decisions could not be communicated. The next logical step is to introduce this ability by appropriately specialising *sense* and *hysteretic-behaviour* functions – analogously to the extension made to the *Hysteretic* agent.

An architecture of an *Extended Task-Oriented* agent $\boldsymbol{A_{ETO}}$ (a sub-class of the *Task-Oriented* agent) contains two additional communication components *request* and *accept*, and is defined as a tuple

<***C***, ***S***, ***E, I***, ***T***, *sense, tropistic-behaviour, hysteretic-behaviour, update, decision, combination, notify, listen,* ***request, accept***, *response*>

The *accept* function is specialised from the *sense* function, and *request* function is a specialised *hysteretic-behaviour*. The *Extended Task-Oriented* agent extends the *Task-Oriented* agent in the same way as the *Extended Hysteretic* agent extends the *Hysteretic* agent. However, new specialised functions deal with communicated tasks rather than behaviour instantiations. In other words, the *accept* function senses that an incoming message contains a *task*, using a suitable sensor, Told: ***t***, and the *request* function arranges to send a *task* to another situated agent by activating a suitable effector, Tell(g, ***t***), where ***t*** is a task, and g is a name of a receiving agent. For example,

**if** [HasBall ∧ CentrePath: blocked] **and**
[SeePartner: (n, a, b, c, d, e) ∧ Near(n) ∧ Free(n)]
**then** Tell(NameOf(n), *task*(*wall_pass*))

**if** [LookingForBall] **and** [Told: *task*(*wall_pass*)]
**then** *task*(*wall_pass*)

The effector Tell(g, ***t***) has to be encoded in a *response*(Tell(g, ***t***)) as any other effector, and the sensory input Told: ***t*** must be decoded by sense(***c***) as any other sensory input. On accepting a *task*, an agent modifies the task state ***T*** accordingly.

Every *hysteretic-behaviour* instantiation (a rule) is applicable if it is activated by a current task and may require from other actions to be not in progress. It is worth noting a single rule may be a part of different tasks, and therefore may be qualified by different progressing actions. For instance, actions $\boldsymbol{e_2}$ and $\boldsymbol{e_3}$ may potentially qualify the rule (***i***, ***s***, $\boldsymbol{e_1}$), given a current task $\mathbf{t_A}$; and actions $\boldsymbol{e_3}$ and $\boldsymbol{e_4}$ are possible qualifications for the same rule when task $\mathbf{t_B}$ is chosen. So the union {$\boldsymbol{e_2}$ , $\boldsymbol{e_3}$ , $\boldsymbol{e_4}$} is the minimal qualification set for the rule (***i***, ***s***, $\boldsymbol{e_1}$). Generally, a qualification set for a rule (***i***, ***s***, ***e***) includes the action ***e*** itself – to avoid self-interruptions.

## 3.2 Process-Oriented Agents

The *Extended Task-Oriented* agent is capable of performing certain tactical elements in real-time by activating only a subset of its behaviour functions, and thus concentrating only on a specified task, possibly with some assistance from the agent(s) which *accept*-ed the *request*. 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.

We intend to introduce a new notion, a *process*, to capture this kind of coherent behaviour - when an agent is engaged in an activity requiring several tasks. Although planning is much simpler in the task-space, and it is quite natural to assume that a process may be activated by a plan leading to a goal, we do not restrict a process-orientation to be a result of pure deliberation. In other words, a process may just constrain 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-run, wall-pass, 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 $\boldsymbol{A_{PO}}$

<***C***, ***S***, ***E, I***, ***T***, ***P***, *sense, tropistic-behaviour, hysteretic-behaviour, update, decision, combination,* ***engage***, ***tactics***, *notify, listen, request, accept, response*>

A *Process-Oriented* agent maintains a process state ***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 a particular internal state, sensory inputs, current task and current process (figure 1).

An explicit inclusion of yet another function, *process-oriented-behaviour* $\boldsymbol{I} \times \boldsymbol{S} \times \boldsymbol{T} \times \boldsymbol{P} \rightarrow \boldsymbol{E}$ extends the architecture $\boldsymbol{A_{PO}}$ towards a subsumption-style architecture. Analogously to previously described subsumption dependencies, possible conflicts between behaviour instantiations $\boldsymbol{e_1}$ = *tropistic-behaviour*(***s***), $\boldsymbol{e_2}$ = *hysteretic-behaviour*(***i***, ***s***), $\boldsymbol{e_3}$ = *task-oriented-behaviour*(***i***, ***s***, ***t***), and $\boldsymbol{e_4}$ = *process-oriented-behaviour*(***i***, ***s***, ***t***, ***p***), triggered by the same sensory input ***s,*** the internal state ***i***, and task state ***e***, can be resolved according to the process state ***p*** - again potentially leading to inhibition of the simpler level(s) behaviour.

The process-orientation and coherent task switching become critical for efficient multi-agent teamwork. A *Process-Oriented* agent is capable of consolidating related tasks into coherent processes. However, when a process is divided among agents, the interface must be clearly defined. Architecture of an *Extended Process-Oriented* agent $\boldsymbol{A_{EPO}}$ (a sub-class of the *Process-Oriented* agent) contains two additional communication components *initiate* and *participate*, and is defined as a tuple

<***C***, ***S***, ***E, I***, ***T***, ***P***, *sense, tropistic-behaviour, hysteretic-behaviour, update, decision, combination, engage, tactics, notify, listen, request, accept,* ***initiate***, ***participate***, *response*>

As before, the communication components are specialised from the *sense* and *hysteretic-behaviour* function, and are intended to deal with communicated and coordinated processes.

Implementation of process-orientation is for the most part analogous to the implementation of task-orientation. One important difference is, however, that all the rules in a set of tasks composing a process cannot be simply represented by their union. Let us say, for example, that a process ***p*** may include task $\mathbf{t_A}$, *combining* rules ($\boldsymbol{i_1}$, $\boldsymbol{s_1}$, $\boldsymbol{e_1}$) and ($\boldsymbol{i_2}$, $\boldsymbol{s_2}$, $\boldsymbol{e_2}$), and task $\mathbf{t_B}$, *combining* rules ($\boldsymbol{i_2}$, $\boldsymbol{s_2}$, $\boldsymbol{e_2}$) and ($\boldsymbol{i_3}$, $\boldsymbol{s_3}$, $\boldsymbol{e_3}$). Then, at any given time, the rules ($\boldsymbol{i_1}$, $\boldsymbol{s_1}$, $\boldsymbol{e_1}$) and ($\boldsymbol{i_3}$, $\boldsymbol{s_3}$, $\boldsymbol{e_3}$) cannot belong to any current task (be it $\mathbf{t_A}$ or $\mathbf{t_B}$), and therefore, are not considered together by the *hysteretic-behaviour* function. Thus, the addition of tactical layers to the agent architecture not only expands the representation, but also increases the complexity of the agent behaviour. In other words, a process-oriented behaviour is potentially more than just a sum of *tropistic*, *hysteretic* and *task-oriented* behaviours, and therefore, its emergent properties cannot be reduced to simple reactions.

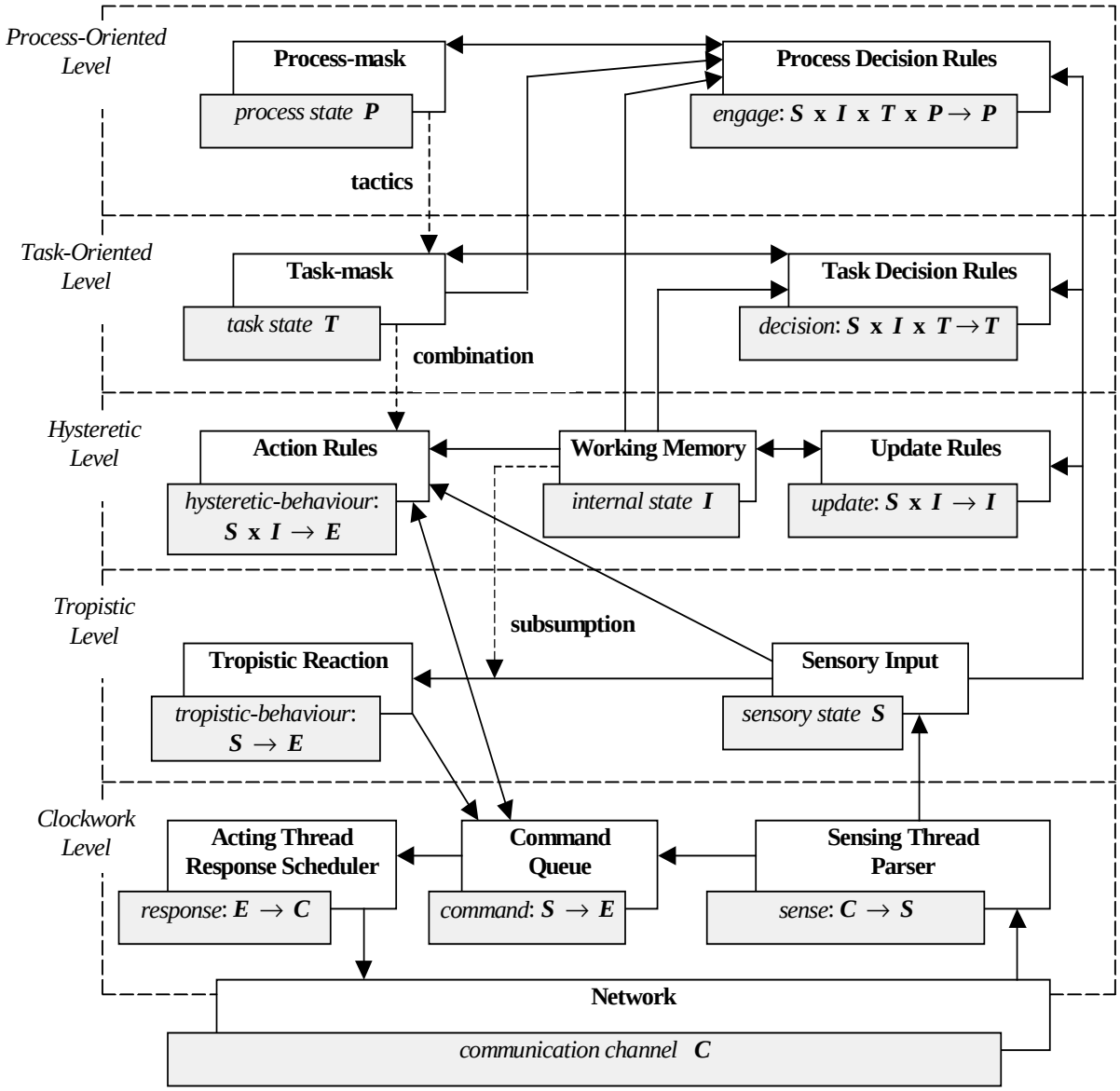


Figure 1. Tactical Agent Architecture

## 4 Tactical Coordination

A *dynamical system* can be characterised as "a system whose state changes over time, and where effects flow forward in time so that the non-input part of the state at one time can only depend on its earlier states" [Sandewall, 1994]. Agents of the system perform actions influencing state variables and changing the system state.

We define a dynamic multi-agent system by a set of architecture types $\boldsymbol{A} \subseteq \{ \boldsymbol{A_S}, \boldsymbol{A_T}, \boldsymbol{A_H}, \boldsymbol{A_{EH}}, \boldsymbol{A_{TO}}, \boldsymbol{A_{ETO}}, \boldsymbol{A_{PO}}, \boldsymbol{A_{EPO}} \}$, and a particular value of a time parameter *t*. Given a finite set of agents $g_k$ ($1 \le k \le N$) instantiated from the architectures in $\boldsymbol{A}$, one can construct a dynamic system $\boldsymbol{V_A}$ (based on $\boldsymbol{A}$) which maps an initial state and a time value to a state. More precisely, $\boldsymbol{V_A}$ is a function $\boldsymbol{U} \times \boldsymbol{R} \rightarrow \boldsymbol{U}$, where $\boldsymbol{U}$ is the set of possible states $\boldsymbol{I_1} \times \boldsymbol{T_1} \times \boldsymbol{P_1} \times \ldots \times \boldsymbol{I_{N-1}} \times \boldsymbol{T_{N-1}} \times \boldsymbol{P_{N-1}} \times \boldsymbol{W}$ and $\boldsymbol{R}$ is the set of real numbers – assuming, without loss of generality, that agent $g_N$ is a *Simulator* agent. We denote a state generated by dynamic system $\boldsymbol{V_A}$ at the time instant *t* as $\boldsymbol{V_A}(t)$.

The proposed hierarchical framework includes a number of implementation classes, allowing us to define heterogeneous teams of autonomous agents participating in a simulated soccer game. 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. The concept of a group enables formal tactical reasoning about multi-agent teamwork. We say that agents $g_k$ ($1 \le k \le M$) instantiated from the architectures in $\boldsymbol{A}$ cooperate in a task group $\boldsymbol{G_T} = \{g_1, \ldots, g_M\}$ if and only if, given a state $\boldsymbol{V_A}(t)$,

$$\forall\, g_i, g_j \in \boldsymbol{G_T}, \quad \exists\, \mathbf{t_i}(t) \in \boldsymbol{T}, \exists\, \mathbf{t_j}(t) \in \boldsymbol{T}, \text{ such that } \mathbf{t_i}(t) = \mathbf{t_j}(t),$$

where $\mathbf{t_i}(t)$ is the task state of the agent $g_i$ at the time point *t*.

Analogously, agents $g_k$ ($1 \le k \le M$) instantiated from the architectures in $\boldsymbol{A}$ cooperate in a process group $\boldsymbol{G_P} = \{g_1, \ldots, g_M\}$ if and only if, given a state $\boldsymbol{V_A}(t)$,

$$\forall\, g_i, g_j \in \boldsymbol{G_T}, \quad \exists\, \mathbf{p_i}(t) \in \boldsymbol{P}, \exists\ \mathbf{p_j}(t) \in \boldsymbol{P}, \text{ such that } \mathbf{p_i}(t) = \mathbf{p_j}(t),$$

where $\mathbf{p_i}(t)$ is the process state of the agent $g_i$ at the time point *t*.

It is important to realise that a teamwork is formalised on the system level. In other words, even if a tactical agent is engaged in group tasks and/or processes, the overall team behaviour/tactics emerges only as a result of agent interactions.

## 5 Action Theories and Systematic Models

Ideally, a conversion from a domain described as a logic theory of actions into a behaviour-based multi-agent system should preserve the meaning of the domain description. In other words, state transitions produced by behaviours of autonomous agents must be warranted by logic-based reasoning. It is possible to consider a generic class of *systematic models* $<\boldsymbol{Th}, \boldsymbol{Tr}, \boldsymbol{V}>$, where each instance of an action theory $\boldsymbol{Th}$ provides a validation criterion for a dynamic system $\boldsymbol{V}$, and the translation $\boldsymbol{Tr}$: $\boldsymbol{Th} \Rightarrow \boldsymbol{V}$ is sound with respect to reasoning warranted by the theory $\boldsymbol{Th}$ [Prokopenko, 1999].

In this section, we use a basic action theory describing unconstrained domains to derive a dynamic multi-agent system based on the *Hysteretic* agent architecture. Then a more complex class of domains with ramifications is mapped into another dynamic system based on the *Extended Hysteretic* agent architecture. Both translations are sound with respect to the underlying action theories. The approach follows a *narrative time-line approach* of Sandewall [1996] and allows us to define continuous change, discrete discontinuities, actions with duration, composite actions, and the distinction between success and failure of an action. We will adopt from [Sandewall, 1996] the following notation:

**H**(t, f:v): fluent f has the value v at time t;
**X**(t,f): fluent f is exempt from minimisation of discontinuities (the occlusion operator);
**G**(s, *a*): the action *a* is invoked at time s;
**A**(s, *a*): the action *a* is applicable at time s;
$\mathbf{D_s}$([s,t], *a*): the action *a* is *successfully executed* over the time interval [s,t];
$\mathbf{D_f}$([s,t], *a*): the action *a* *fails* over the time interval [s,t];
$\mathbf{D_c}$([s,t], *a*): the action *a* is *being executed* during the interval [s,t).

### 5.1 Action Theories for Reactive Planning

We start with simple deterministic artificial worlds where domain constraints are not defined, and therefore all action effects are direct. All initially given formulae **H**(t, f:v) will be called observation descriptions, and all initially given formulae **G**(t, *a*) will be referred to as plan descriptions.

The action *success* description has the following form:

$$\mathbf{D_s}([s,t], a) \rightarrow \mathbf{H}(t, \omega_a) \qquad (1)$$

where $\omega_a$ is the post-condition of the action *a* given at the termination time [Sandewall, 1996]. For example,

$$\mathbf{D_s}([s,t], \mathit{PASS}(x, y, p)) \rightarrow \mathbf{H}(t, \text{possession}(y))$$

describes successful execution of the *PASS* action, applicability description of which can be given as

$$\mathbf{A}(s, \mathit{PASS}(x, y, p)) \leftrightarrow \mathbf{H}(s, \text{possession}(x)) \wedge \mathbf{H}(s, \text{free}(y)) \wedge \mathbf{H}(s, \text{sustains}(x, p))$$

An action, once invoked, continues towards a success at which instant it terminates (unless there is a qualification that forces it to fail earlier):

$\mathbf{A}$(s, *PASS*(x, y, p)) $\wedge$ $\mathbf{G}$(s, *PASS*(x, y, p)) $\rightarrow$
$\mathbf{H}$(s, ball_velocity(x):= p)

$\mathbf{H}$(t, see_ball(x): z) $\wedge$ $\mathbf{H}$(t, see_partner(x): y) $\wedge$
$\mathbf{H}$(t, very_near(y): z) $\wedge$ $\mathbf{D_c}$([s,t], *PASS*(x, y, p)) $\rightarrow$
$\mathbf{D_s}$([s,t], *PASS*(x, y, p))

Axioms so exemplified are called *invocation* and *termination* descriptions respectively:

$\mathbf{A}(s, a) \wedge \mathbf{G}(s, a) \rightarrow \mathbf{H}(s, \gamma_a)$ (2)
$\mu_i \wedge \mathbf{D_c}([s,t], a) \rightarrow \mathbf{D_s}([s,t], a)$ (3)

where $\gamma_a$ is the *invocation* condition and $\mu_i$ is one of the *termination* conditions.

An action failure is defined by a *failure* description and by a *failure effects* description:

$\delta_i \wedge \mathbf{D_c}([s,t], a) \wedge \neg\mathbf{D_s}([s,t], a) \rightarrow \mathbf{D_f}([s,t], a)$ (4)
$\mathbf{D_c}([s,t], a) \wedge \mathbf{D_f}([s,t], a) \rightarrow \mathbf{H}(t, \tau_a)$ (5)

where $\tau_a$ is the *failure post-condition*.

We will denote the described theory of actions as $\boldsymbol{Th_1} = \langle \boldsymbol{D}, \boldsymbol{M} \rangle$, where all domain axioms compose $\boldsymbol{D}$, and $\boldsymbol{M}$ is a specific minimisation policy.

It can be shown that the described theory of actions $\boldsymbol{Th_1} = \langle \boldsymbol{D}, \boldsymbol{M} \rangle$ provides a validation criterion for a dynamic multi-agent system $\boldsymbol{V_{S_H}}$ based on the *Simulator* and *Hysteretic* agent architectures $\boldsymbol{S_H} = \{\boldsymbol{A_S}, \boldsymbol{A_H}\}$. Although the time in $\boldsymbol{Th_1}$ is continuous we can, nevertheless, validate an atomic formula tr(f):v in $\boldsymbol{V_{S_H}}$ ($t$) at the time instant $t$ if $\mathbf{H}(t,$ f:v) is entailed by a consistent theory $\boldsymbol{Th_1}$. In other words, the translation $\boldsymbol{Tr_1}$: $\boldsymbol{Th_1} \Rightarrow \boldsymbol{V_{S_H}}$ is sound:

**Proposition 1.** Given a deterministic unconstrained domain $\boldsymbol{D}$ described by a consistent action theory, there exists an assignment relation $\boldsymbol{P} \subseteq \boldsymbol{G} \times \boldsymbol{D}$ for a set of agent class names $\boldsymbol{G}$, such that the translation $\boldsymbol{Tr_1}$: $\boldsymbol{D} \Rightarrow \boldsymbol{V_{S_H}}$ produces a dynamic system (based on $\{\boldsymbol{A_S}, \boldsymbol{A_H}\}$) where all atomic formulae are valid.

## 5.2 Ramifications

An extended action theory allows us to reason about ramifications and interactions. Typically, indirect changes (ramifications) are non-monotonically derived as consequences of domain constraints. For example,

$\mathbf{H}$(t, near(x): y ) $\wedge$ $\mathbf{H}$(t, near(x): z) $\rightarrow$
$\mathbf{H}$(t, near(y) = z) $\wedge$ $\mathbf{X}$(t, near(y) = z)

This reassignment constraint uses the occlusion operator $\mathbf{X}$(t,f) and excludes the indirect effects from the law of inertia. This effectively specifies the direction of the dependency and makes the latter look like a "causal rule" producing necessary ramifications (McCain and Turner 1995, Gustaffson and Doherty 1996).

Another form of ramifications describes an *interaction* when one continuous action triggers another:

$\lambda_i \wedge \mathbf{D_c}([s,t], a) \wedge \neg\mathbf{D_f}([s,t], a) \rightarrow \mathbf{G}(t, b)$ (6)

where each $\lambda_i$ represents an interaction condition, and $b$ is another action invoked by occurrences of $\lambda_i$ during the execution of the action $a$. For example,

$\mathbf{H}$(t, see_opponent(x): z) $\wedge$ $\mathbf{H}$(t, near(x): z) $\wedge$ $\mathbf{H}$(t, see_partner(x): y) $\wedge$ $\mathbf{D_c}$([s,t], *DRIBBLE*(x, d) ) $\wedge$ $\neg\mathbf{D_f}$([s,t], *DRIBBLE*(x, d)) $\rightarrow$ $\mathbf{G}$(t, *PASS*(x, y, *distance*(x, y)))

$\mathbf{G}(t, b)$ is the only specified effect of the interaction. Therefore other effects of the action $b$ (defined in its success, failure, and/or interaction descriptions) can be viewed as ramifications of this interaction. They do not have to be specified explicitly with every such interaction and are supposed to be implied indirectly. Possible preconditions for the action invocation are checked by the applicability description $\mathbf{A}(t, b)$. In general, any expression of the form

$\lambda_i \rightarrow \mathbf{G}(t, b)$ (7)

can be considered as a (trivial) *interaction* description.

We will denote the described theory of actions as $\boldsymbol{T_2} = \langle \boldsymbol{D}, \boldsymbol{M} \rangle$, where domain axioms composing $\boldsymbol{D}$ may include domain constraints and interaction descriptions.

The Reasoning about Action tradition proposes to use domain constraints and/or causal laws separated from action specifications in order to derive indirect effects of an action. In a multi-agent framework, a similar solution can be achieved by embedding indirect effects in situated behaviours of autonomous reactive agents.

An extended translation $\boldsymbol{Tr_2}$: $\boldsymbol{Th_2} \Rightarrow \boldsymbol{V_{S_EH}}$ from a domain described in the theory of actions $\boldsymbol{T_2}$ to a dynamic system based on the *Simulator* and *Extended Hysteretic* agent architectures $\boldsymbol{S_EH} = \{\boldsymbol{A_S}, \boldsymbol{A_{EH}}\}$ uniformly translates domain constraints and interaction descriptions into situated behaviours, thus allowing to account for possible ramifications. The extended theory of actions $\boldsymbol{Th_2} = \langle \boldsymbol{D}, \boldsymbol{M} \rangle$ provides a validation criterion for the system $\boldsymbol{V_{S_EH}}$, and it can be shown that the translation $\boldsymbol{Tr_2}$: $\boldsymbol{Th_2} \Rightarrow \boldsymbol{V_{S_EH}}$ is sound.

**Proposition 2.** Given a deterministic domain $\boldsymbol{D}$ described by a consistent action theory, there exists an assignment relation $\boldsymbol{P} \subseteq \boldsymbol{G} \times \boldsymbol{D}$ for a set of agent class names $\boldsymbol{G}$, such that the translation $\boldsymbol{Tr_2}$: $\boldsymbol{D} \Rightarrow \boldsymbol{V_{S_EH}}$ produces a dynamic system (based on $\{\boldsymbol{A_S}, \boldsymbol{A_{EH}}\}$) where all atomic formulae are valid.

The translation $\boldsymbol{Tr_2}$ may produce a multi-agent system based on the *Simulator* and *Hysteretic* agent architectures $\boldsymbol{S_H} = \{\boldsymbol{A_S}, \boldsymbol{A_H}\}$. It occurs when all messages are communicated internally within an agent. In other words, interactions and domain constraints are defined in terms of internal variables. The class of action do-

mains where the translation yields these particularly simple results is the class of domains with local ramifications.

Furthermore, if a deterministic domain with local ramifications is described only by plan and trivial interaction descriptions (7), then its translation produces a multi-agent system based on the *Simulator* and *Tropistic* agent architectures $\boldsymbol{S_T} = \{\boldsymbol{A_S}, \boldsymbol{A_T}\}$. The *sense* function in $\boldsymbol{A_T}$ captures all trivial interaction conditions, the *tropistic-behaviour* implements all trivial interaction descriptions, and the *command* invokes all (pre-)planned actions, producing timed *response*.

## 6 Conclusions

The view taken in this paper is that, rather than defining situated or tactical reasoning *ad hoc*, it is desirable to categorise agents according to their functionality and reactions to the environment, and identify corresponding classes of action theories. Then the reasoning exhibited by agents of a certain type and validated by particular action theories can be declared to be situated, tactical, social and so on. In other words, the principal target is a description of systematic models underlying increasing levels of agent reasoning abilities. The reported results demonstrated that this is achievable at the situated level.

Preliminary results on the systematic models for *Task-Oriented* and *Process-Oriented* behaviour were obtained as well. In particular, the action theory for process-orientation captures a goal-oriented behaviour described in [Sandewall, 1997]. Unlike this approach, where both descriptions are expressed in logic, we do not require that the agent architecture derived from a higher-level representation is a logic-based formalism. Another important distinction from the Sandewall formalism [1997] is that goal-directed behaviour is characterised via appropriate task orientation rather than as a process carrying out a plan.

The described hierarchical framework has been realised in the RoboCup Simulation League domain (team Cyberoos). Not only did it provide a solid design approach to object-orientation, but it also enabled rigorous incremental implementation and testing of software agents and their modules. In particular, the framework allowed us to correlate enhancements in agent architecture with tangible improvements in team performance. The Cyberoos'98 team took third place in the 1998 Pacific Rim RoboCup competition. Cyberoos'99 finished in the top half of the RoboCup-1999, winning five games out of ten.

## References


[Baral and Son, 1996] Baral, C., and Son, T. Relating Theories of Actions and Reactive Robot Control. In Proceedings of the AAAI 1996 Workshop on Theories of Action and Planning: Bridging the Gap. Portland 1996.

[Brooks, 1991] Brooks, R.A. Intelligence Without Reason. In Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, 569-595 Morgan Kaufmann, 1991.

[Genesereth and Nilsson, 1987] Genesereth, M.R., and Nilsson, N.J. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, 1987.

[Gustaffson and Doherty, 1996] Gustafsson, J. and Doherty, P. Embracing occlusion in specifying the indirect effects of actions. In L. Aiello, J. Doyle, and S. Shapiro (editors), Proceedings of the Fifth International Conference on Knowledge Representation and Reasoning. Morgan-Kaufmann, 1996.

[Kaelbling and Rosenchein, 1990] 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*, 35-48, MIT/Elsevier, 1990.

[Kitano, *et al*, 1997] Kitano, H., Tambe, M., Stone, P., Veloso, M., Coradeschi, S., Osawa, E., Matsubara, H., Noda, I., and Asada, M. The RoboCup Synthetic Agent Challenge. In Proceedings of the 15th Int'l Joint Conference on Artificial Intelligence, 1997.

[McCain and Turner, 1995] McCain, N., and Turner, H. A Causal Theory of Ramifications and Qualifications. In Proceedings of the 14th International Joint Conference on Artificial Intelligence, Montreal 1995, 1978-1984.

[Parsons *et al*, 1998] Parsons, S., Sierra, C., and Jennings, N. Multi-context Argumentative Agents. In Proceedings of the Fourth International Symposium on Logical Formalizations of Commonsense Reasoning 1998.

[Prokopenko and Jauregui, 1997] 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.

[Prokopenko, *et al*., 1998] Prokopenko, M., Kowalczyk R., Lee M., Wong, W.-Y. 1998. Designing and Modelling Situated Agents Systematically: Cyberoos'98. In Proceedings of the PRICAI-98 Workshop on RoboCup, 75-89. Singapore 1998.

[Prokopenko, 1999] Prokopenko, M. Situated Reasoning in Multi-Agent Systems. In AAAI Technical Report SS-99-05, the AAAI-99 Spring Symposium on Hybrid Systems and AI, 158 - 163. Stanford 1999.

[Sandewall, 1994] Sandewall, E. *Features and Fluents. The Representation of Knowledge about Dynamical Systems. Volume I.* Oxford University Press 1994.

[Sandewall, 1996] Sandewall, E. Towards the Validation of High-level Action Descriptions from their Low-level Definitions. *Linköping electronic articles in Computer and Information science*, (1):4 1996.

[Sandewall, 1997] Sandewall, E. Logic-based Modelling of Goal-Directed Behaviour. *Linköping electronic articles in Computer and Information science*, (2):19 1997.

[Steels, 1990] Steels, L. Exploiting Analogical Representations. In Maes, P. (ed) *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, 71-88. MIT/Elsevier, 1990.