On Situated Reasoning in Multi-Agent Systems

Mikhail Prokopenko

Applied Artificial Intelligence Project
CSIRO Mathematical and Information Sciences
Locked Bag 17, North Ryde, NSW 1670, Australia
mikhail.prokopenko@cmis.csiro.au

Abstract

In this paper we aim to analyse relationships between different approaches to formalising interactivity in dynamic systems. 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 Multi-agent Systems. It usually considers autonomous agents reacting to changes in external environment and (ideally) exhibiting emergent behaviour. We first attempt to formally define various types of situated agent architectures and encapsulate them in a hierarchical framework. We then analyse and identify domain classes and action theories corresponding to given agent architecture types. This approach can specifically assist in mapping logic theories of actions to reactive agent architectures, where ramifications are embedded in situated behaviours. The described hierarchical framework has been used in the RoboCup Simulation League domain, resulting in implementation of the Cyberoos'98 – a heterogeneous soccer team of autonomous software agents (3rd place winner of the Pacific Rim series at PRICAI-98).

Introduction

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).

The idea that reactive behaviours can be proved to be correct with respect to a theory of actions (and in some cases can be derived from it) is relatively new. For instance, a connection between theories of actions and reactive robot control architectures based on the paradigm of situated activity is explored in (Baral and Son 1996). This approach formalises further the concept of "an action leading to a goal" defined at the representation level in the *situated automata approach* (Kaelbling and Rosenschein 1990) and follows the latter in relating declarative agent specifications and situated behaviours. Recently the related problem of "formally proving high-level effect descriptions of actions from low-level operational definitions" (Sandewall 1996) was addressed in the context of robotic knowledge validation. Unlike this approach, where both descriptions are expressed in logic, we do not require from an agent architecture derived from a higher-level representation to be a logic-based formalism. On the contrary, the resulting architecture may contain only reactive behaviours validated with respect to a meta-level action theory (Prokopenko and Jauregui 1997, Prokopenko et al 1998). We must stress that our approach aims not only at obtaining sound translation procedures but also (and more importantly) at analysing and identifying classes of domains corresponding to certain types of agent architectures.

More precisely, an attempt is made to select a certain class of domains and describe a procedure mapping a domain description (given as a logic theory of actions) into a behaviour-based multi-agent system. Such a conversion should ideally preserve the meaning of the domain description as compared with the multi-agent system's dynamics. In other words, state transitions produced by behaviours of autonomous agents must be warranted by logic-based reasoning about actions and change.

Initially, we describe a hierarchical framework for situated agent architectures. Then a basic action theory describing unconstrained domains is used to derive a dynamic multi-agent system based on a simple reactive agent architecture. A more complex class of domains with logical and causal constraints is mapped into another dynamic system (based on an extended architecture), using an augmented translation procedure. Both translations are shown sound with respect to underlying action theories.

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). We first attempt to incorporate these results in a framework suitable for situated synthetic agents. Then we try to extend the architecture, while retaining the rigour and clarity of fundamental definitions.

Environment Simulator

We define a *Simulator* agent as a tuple A_s

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

where W is a set of all external states, P is a set of all possible partitions of W, A is a set of situated agents, E is a set of effectors, and 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*: $A \rightarrow P$.

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 A. In other words, the *Simulator* projects an external state and a situated agent into an element of the viewable partition of external states, by using *projection*: $W \times A \rightarrow 2^W$, where 2^W is the power-set of W. The exact range of the *projection* function is the external states partition selected by *view* from the set P of all possible partitions of W. More precisely,

$$\forall w \in W, \forall a \in A, \ \ projection(w, a) \in view(a)$$

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

Tropistic Agents

Having defined the architecture of the *Simulator* agent, we formally describe an *Abstract Tropistic* agent as a tuple A_{AT}

$$<C, S, E, sense, tropistic\text{-}behaviour, response>,$$

where S is a set of agent sensory states, and C and E denote the same components as before. The sensory function is defined as *sense*: $C \rightarrow S$, where an element of C is expected to carry the information on $projection(w, a)$. Activity of the agent is characterised by *tropistic-behaviour*: $S \rightarrow E$. We do not intend here to formally define the notion of reactive planning. However, by allowing the set E to include composite effectors $e_1; e_2$, where $e_1 \in E$, $e_2 \in E$, we can implicitly account for the case of tropistic planning - when a situated agent reacts to stimuli S with an n-length sequence of effectors. The *response* function takes care of communicating the selected behaviour to the *Simulator* by encoding *response*: $E \rightarrow C$. This abstract class may not have any instances because the *tropistic-behaviour* function is not implemented at this level of the hierarchy.

It is interesting at this stage to consider a very simple sub-class of the *Abstract Tropistic* agent - a *Clockwork* agent. This class has a specialised sensory function *timer*: $C \rightarrow S$, and does not specialise the function *sense* in any other way. In other words, a *Clockwork* agent is able to distinguish only between external states with different time values, having no other sensors apart from the *timer*. In addition, this class specialises the *tropistic-behaviour* function by introducing the *command* function defined as *command*: $S \rightarrow E$. Since the only sensor available at this level is the *timer*, the agent behaviour is predefined and is totally driven by time values. In other words, like a clockwork mechanism, a *Clockwork* agent executes its fixed behaviour as a sequence of commands sent to the *Simulator* at predefined time points. Formally, the *Clockwork* agent class is defined as a tuple A_{CW}

$$<C, S, E, sense, \mathbf{timer}, tropistic\text{-}behaviour, \mathbf{command}, response>,$$

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

The *Tropistic* agent class A_T is derived from the *Clockwork* agent and finally allows us to implement the *tropistic-behaviour* function.

In practice, it is almost impossible to express each instantiation (e, s) of the *tropistic-behaviour* function $e = tropistic\text{-}behaviour(s)$ in terms of complete sensory states. Instead, we represent such behaviour instantiations in terms of partial sensory states. For example, the following rules, given in the form similar to control rules (Baral and Son 1996) or condition-action pairs (Kaelbling and Rosenschein 1990), describe 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 S and has to be evaluated as true in order to activate effector(s) on the right-hand side. A sentence α in this component specifies the set of states from S consistent with α. 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.

Hysteretic Agents

A *Hysteretic* agent is defined here as a reactive agent maintaining internal state I and using it as well as sensory states S in activating effectors E; i.e. its activity is characterised by *hysteretic-behaviour*: $I \times S \rightarrow E$. Again, we allow the set E to include composite effectors $e_1; e_2$, where $e_1 \in E$, $e_2 \in E$, 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*: $I \times S \rightarrow I$. A *Hysteretic* agent reacts to stimuli s sensed by $sense(c)$ and activates effectors e according to $hysteretic\text{-}behaviour(i, s)$. The agent neither has full knowledge about the state $do(e, w)$ 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 A_H

<***C, S, E, I,*** *sense, timer, tropistic-behaviour, command,* ***hysteretic-behaviour, update,*** *response*>

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 ***I*** and ***S*** respectively.

An *Extended Hysteretic* agent 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, timer, tropistic-behaviour, command, hysteretic-behaviour,* ***notify, listen,*** *update, 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

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

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 [SeePartner: (n, d, angle) $\wedge$ SeeBall: (dist, dir) $\wedge$ 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.

Dynamic multi-agent systems

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). The 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 $\boldsymbol{A} \subseteq \{ A_S, A_{CW}, A_T, A_H, A_{EH} \}$, and a particular value of a time parameter $\boldsymbol{t}$. 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 V_A (based on $\boldsymbol{A}$) which maps an initial state and a time value to a state.

More precisely, V_A is a function $\boldsymbol{U} \times \boldsymbol{R} \rightarrow \boldsymbol{U}$, where $\boldsymbol{U}$ is the set of possible states $I_1 \times \ldots \times I_{N-1} \times W$ 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 the dynamic system V_A at the time instant t as $V_A(t)$.

Action Theories for Situated Reasoning

The approach to representing operational definitions and effect descriptions of continuous actions (Sandewall 1996) follows a *narrative time-line approach* 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).

The set of axioms in (Sandewall 1996) specifies properties and relationships of these predicates. All state variables (fluents) in a described domain may have an argument. We assume that multi-argument fluents can be reified and alternatively represented by unary fluents without an expressibility loss. In order to declare that a fluent does not have a value we use the *nil* symbol: **H**(t, f:*nil*) abbreviates $\neg\exists$v [**H**(t, f:v)]. Another syntactic sugar is introduced for anonymous variables: **H**(t, f:_) abbreviates $\exists$v [**H**(t, f:v)], assuming that the sentence where the Skolem constant replaced a variable, has had no two quantifiers referring to the same variable v. By definition, **H**(t, f:*nil*) $\leftrightarrow \neg$**H**(t, f:_). The reassignment operator := will be used to abbreviate **H**(t,f:v) $\wedge$ **X**(t,f) as **H**(t, f:=v).

Basic Action Theory for Unconstrained Domains

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:

$$\mathbf{D}_s([s,t], a) \rightarrow \mathbf{H}(t, \omega_a) \tag{1}$$

where ω_a is the post-condition of the action *a* given at the termination time (Sandewall 1996). For example,

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

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

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

An action, once invoked, continues towards a success at which instant it terminates (unless there is a qualification that forces it to fail earlier). The *invocation* and *termination* descriptions respectively are given as follows:

$$\mathbf{A}(s, a) \wedge \mathbf{G}(s, a) \rightarrow \mathbf{H}(s, \gamma_a) \tag{2}$$
$$\mu_i \wedge \mathbf{D}_c([s,t], a) \rightarrow \mathbf{D}_s([s,t], a) \tag{3}$$

where γ_a is the *invocation* condition and μ_i is one of the *termination* conditions.

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

$$\delta_i \wedge \mathbf{D}_c([s,t], a) \wedge \neg\mathbf{D}_s([s,t], a) \rightarrow \mathbf{D}_f([s,t], a) \tag{4}$$
$$\mathbf{D}_c([s,t], a) \wedge \mathbf{D}_f([s,t], a) \rightarrow \mathbf{H}(t, \tau_a) \tag{5}$$

where τ_a is the *failure post-condition.*

We will denote the described theory of actions as $\boldsymbol{T_I}$ = <***D***, ***M***>, where all domain axioms compose ***D***, and ***M*** is a specific minimisation policy. The chronological minimisation of discontinuities in piecewise continuous fluents should, in general, be complemented by maximisation of action duration (Sandewall 1996).

Basic Translation

In this section we employ a basic translation $\boldsymbol{Tr_I}$: ***D*** $\Rightarrow$ $\boldsymbol{V_{S\_H}}$ from a domain described by the theory of actions $\boldsymbol{T_I}$ to a dynamic multi-agent system based on the *Simulator* and *Hysteretic* agent architectures ***S_H*** = {$\boldsymbol{A_S}$, $\boldsymbol{A_H}$}.

We introduce a set ***G*** of agent sub-classes (derived from elements of ***S_H***) and assign all domain descriptions in ***D*** to corresponding classes. Formally, an assignment relation $\boldsymbol{P} \subseteq \boldsymbol{G} \times \boldsymbol{D}$ is defined such that $\forall\, d \in \boldsymbol{D}, \exists\, g \in \boldsymbol{G}, (g, d) \in \boldsymbol{P}$. For all agent classes $g \in \boldsymbol{G}$, the translation $\boldsymbol{Tr_I}$ introduces appropriate sensor, effector and internal variables names tr(f), and processes all (assigned to *g*) descriptions *d* such that $(g, d) \in \boldsymbol{P}$. In particular, the following steps are performed:

- for all plan descriptions **G**(t, *a*) produce a *behaviour* instantiation
  if [tr(φ_a)] **and** [Timer: t] **then** *start_a*
  where φ_a is the applicability condition of action *a*, Timer is a sensor, t denotes a current time reading; the effector *start_a* executes the translated invocation condition tr(γ_a) (2) and initiates the Started_*a* variable as Started_*a*: t;
- for all observation descriptions **H**(t, f:v) produce an *update* and/or *sense* instantiation(s)
  if [Timer: t] **then** {tr(f): v};
- for all termination descriptions (3) produce a *behaviour* instantiation
  if [Started_*a*: s $\wedge$ tr($\neg\omega_a$)] **and** [tr(μ_i) $\wedge$ Timer: t] **then** *stop_a*
  where *a* is the action in $\mathbf{D}_s$([s,t], *a*), ω_a is its post-condition; the effector *stop_a* executes the translated success condition tr(ω_a) (1) and sets the Started_*a* variable as Started_*a*: nil;
- for all failure descriptions (4) produce a *behaviour* instantiation
  if [Started_*a*: s $\wedge$ $\neg$tr($\omega_a \vee \tau_a$)] **and** [tr(δ_i) $\wedge$ Timer: t] **then** *halt_a*
  where *a* is the action in D_f([s,t], *a*); ω_a and τ_a are its post-condition and failure post-condition respectively; the effector *halt_a* executes the translated failure effects description tr(τ_a) (5) and sets the Started_*a* variable as Started_*a*: nil.

The described theory of actions $\boldsymbol{T_I}$ = <***D***, ***M***> provides a validation criterion for the dynamic system $\boldsymbol{V_{S\_H}}$. Although the time in $\boldsymbol{T_I}$ is continuous we can, nevertheless, validate all non-auxiliary atomic formulae tr(f):v in $\boldsymbol{V_{S\_H}}$ (*t*) at the time instant *t* if **H**(*t*, f:v) is entailed by a consistent theory $\boldsymbol{T_I}$. An auxiliary formula Started_*a*: s (s $\neq$ nil) is valid in $\boldsymbol{V_{S\_H}}$ (*t*) if s < *t* $\wedge$ $\mathbf{D}_c$([s,*t*], *a*) is entailed by a consistent theory $\boldsymbol{T_I}$. Similarly, an auxiliary formula Started_*a*: nil is valid in $\boldsymbol{V_{S\_H}}$ (*t*) if a consistent theory $\boldsymbol{T_I}$ entails $\exists$s [s < *t* $\wedge$ ($\mathbf{D}_s$([s,*t*], *a*) $\vee$ $\mathbf{D}_f$([s,*t*], *a*))]. It is easy to verify that the following soundness proposition is true.

Proposition 1. Given a deterministic unconstrained domain ***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 ***G***, such that the translation $\boldsymbol{Tr_I}$: ***D*** $\Rightarrow$ $\boldsymbol{V_{S\_H}}$ produces a dynamic system (based on {$\boldsymbol{A_S}$, $\boldsymbol{A_H}$}) where all atomic formulae are valid.

It follows immediately that a consistent action theory describing a (trivial) domain with only plan descriptions can provide a validation criterion for a dynamic multi-agent system V_{S_CW} based on the *Simulator* and *Clockwork* agent architecture $S\_CW = \{A_S, A_{CW}\}$:

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

Extended Action Theory

The 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,

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

This reassignment constraint uses the occlusion operator **X**(t,f) and excludes (releases) 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:

$$\lambda_i \wedge \mathbf{D}_c([s,t], a) \wedge \neg\mathbf{D}_f([s,t], a) \rightarrow \mathbf{G}(t, b) \qquad (6)$$

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

$\mathbf{H}(t, \text{see\_opponent}(x){:}\ z) \wedge \mathbf{H}(t, \text{near}(x){:}\ z) \wedge \mathbf{H}(t, \text{see\_partner}(x){:}\ y) \wedge \mathbf{D}_c([s,t], \mathit{DRIBBLE}(x, d)) \wedge \neg\mathbf{D}_f([s,t], \mathit{DRIBBLE}(x, d)) \rightarrow \mathbf{G}(t, \mathit{PASS}(x, y, \mathit{distance}(x, y)))$

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 **A**(t, *b*). In general, any expression of the form

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

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

Thus at least two ways to address the ramification problem in a logic characterising piecewise continuous change can be observed: by defining constraints and by specifying *interaction* descriptions for continuous actions.

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

Extended Translation

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 Tr_2: $D \Rightarrow V_{S\_EH}$ from a domain described in the theory of actions T_2 to a dynamic system based on the *Simulator* and *Extended Hysteretic* agent architectures $S\_EH = \{A_S, A_{EH}\}$ translates every domain constraint into a number of reassignment (causal) constraints and introduces additional required names. In particular, it adds the following steps to the translation Tr_1:

- for all interaction descriptions (6) produce a *notify behaviour* instantiation
  if [Started_*a*: s $\wedge$ tr($\neg\tau_a$)] **and** [tr(λ_i) $\wedge$ Timer: t]
  then Tell(g_b, *b_start*),
  and a *listen behaviour* instantiation for an agent g_b
  if [tr(φ_b)] **and** [Told(*b*) $\wedge$ Timer: t] **then** *b_start*
  where *b* is the action in **G**(t, *b*), φ_b is its applicability condition, *a* is the action in $\mathbf{D}_c$([s,t], *a*), τ_a is its failure post-condition, and g_b is the agent receiving the communication;
- for each causal constraint **H**(t, α) $\rightarrow$ **H**(t, f:= v) produce a *notify behaviour* instantiation
  if [tr(α) $\wedge$ $\neg$(tr(f): v)] **and** [Timer: t]
  then Tell($g_{tr(f)}$, *assign_#*)
  and a *listen behaviour* instantiation for an agent $g_{tr(f)}$
  if [Told(*assign_#*) $\wedge$ Timer: t] **then** *assign_#*
  where a (sequentially numbered) *assign_#* effector executes {tr(f): v}, and $g_{tr(f)}$ is the agent receiving the communication.

It is worth noting that domain constraints and interaction descriptions are translated into situated behaviours of the same structure, thus allowing to uniformly embed possible ramifications.

Translation of a definitional domain constraint produces several causal constraints, where a right-hand side fluent is partially defined in terms of the left-hand side. It is well known that some fluents cannot be fully defined in terms of their definitional counterparts. For example, the domain constraint

$$\mathbf{H}(t, \text{free}(x)) \leftrightarrow \neg\exists y\ [\mathbf{H}(t, \text{near}(x){:}\ y)]$$

fully defines the propositional fluent free(x) in terms of the fluent near(x). However, the latter is not uniquely defined in terms of the former. Nevertheless, in order to produce causal constraints, it is sufficient to employ nil and _ values:

$$\mathbf{H}(t, \text{near}(x){:}\ \mathit{nil}) \rightarrow \mathbf{H}(t, \text{free}(x){:=}\ \text{True})$$
$$\mathbf{H}(t, \text{near}(x){:}\ \_) \rightarrow \mathbf{H}(t, \text{free}(x){:=}\ \text{False})$$
$$\mathbf{H}(t, \text{free}(x){:}\ \text{True}) \rightarrow \mathbf{H}(t, \text{near}(x){:=}\ \mathit{nil})$$
$$\mathbf{H}(t, \text{free}(x){:}\ \text{False}) \rightarrow \mathbf{H}(t, \text{near}(x){:=}\ \_)$$

The last constraint leaves open the question what object is near x, but does not justify any unsound formulae.

The extended theory of actions $\boldsymbol{T_2} = \langle \boldsymbol{D}, \boldsymbol{M} \rangle$ provides a validation criterion for the system V_{S_EH}. All atomic formulae tr(f):v in $V_{S\_EH}(t)$ are validated as in $V_{S\_H}(t)$. Analogously, the following soundness proposition is true.

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

Sometimes the translation $\boldsymbol{Tr_2}$ may produce a multi-agent system based on the *Simulator* and *Hysteretic* agent architectures $\boldsymbol{S\_H} = \{A_S, A_H\}$. 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 domains where the translation yields these particularly simple results is the class of domains with local ramifications:

Corollary 2. Given a deterministic domain with local ramifications $\boldsymbol{D}$ described by a consistent action theory, there exists an assignment relation $\boldsymbol{P} \subseteq \boldsymbol{G}$ x $\boldsymbol{D}$ for a set of agent class names $\boldsymbol{G}$, such that the translation $\boldsymbol{Tr_2}$: $\boldsymbol{D} \Rightarrow V_{S\_H}$ produces a dynamic system (based on $\{A_S, A_H\}$) where all atomic formulae are valid.

Furthermore, if a deterministic temporal projection 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 $\boldsymbol{S\_T} = \{A_S, A_T\}$. The *sense* function in 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*. For such trivial domains (described only by plans and trivial interactions) we immediately obtain the following

Corollary 3. Given a trivial deterministic domain with trivial local ramifications $\boldsymbol{D}$ described by a consistent action theory, there exists an assignment relation $\boldsymbol{P} \subseteq \boldsymbol{G}$ x $\boldsymbol{D}$ for a set of agent class names $\boldsymbol{G}$, such that the translation $\boldsymbol{Tr_2}$: $\boldsymbol{D} \Rightarrow V_{S\_T}$ produces a dynamic system (based on $\{A_S, A_T\}$) where all atomic formulae are valid.

Conclusions

The obtained results can be generalised by translating broader classes of action domains into more complex agent architectures. Ideally, any extended translation $\boldsymbol{Tr_k}$: $\boldsymbol{D} \Rightarrow V_A$ must satisfy the important soundness property: state transitions produced by a dynamic multi-agent system V_A are sound with respect to reasoning warranted by an action theory $\boldsymbol{T_k}$. For instance, action theories capturing goal-oriented behaviour axiomatised in (Sandewall 1997) may be used to validate process-oriented agent architectures.

The intention is to consider a generic class of *systematic models* $\langle \boldsymbol{T}, \boldsymbol{Tr}, \boldsymbol{V} \rangle$, where each instance of an action theory $\boldsymbol{T}$ provides a validation criterion for a dynamic system $\boldsymbol{V}$, and the translation $\boldsymbol{Tr}$ is sound. Such systematic models would support uniform specifications of synthetic agents and facilitate a rigorous comparative analysis of different architectures and their ranges of applicability with respect to provably correct logics.

References

1. Baral, C., and Son, T. 1996. 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.
2. Brooks, R.A. 1991. Intelligence Without Reason. In Proceedings of the 12th International Joint Conference on Artificial Intelligence, 569-595. Morgan Kaufmann.
3. Genesereth, M.R., and Nilsson, N.J. 1987. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann.
4. Gustaffson, J., and Doherty, P. 1996. Embracing Occlusion in Specifying the Indirect Effects of Actions. In Proceedings of the 5th International Conference on Principles of Knowledge Representation and Reasoning. Cambridge.
5. Kaelbling, L. P. and Rosenschein, S. J. 1990. Action and planning in embedded agents. In Maes, P. (ed) *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, 35 – 48, Mass.: MIT/Elsevier.
6. McCain, N., and Turner, H. 1995. A Causal Theory of Ramifications and Qualifications. In Proceedings of the 14th International Joint Conference on Artificial Intelligence, 1978-1984. Montreal.
7. Parsons, S., Sierra, C., and Jennings, N. 1998. Multi-context Argumentative Agents. In Proceedings of the Fourth International Symposium on Logical Formalizations of Commonsense Reasoning.
8. Prokopenko, M., Jauregui, V. 1997. Reasoning about Actions in Virtual Reality. In Proceedings of the IJCAI-97 Workshop on Nonmonotonic Reasoning, Action and Change, 159-171.
9. 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. Singapore.
10. Sandewall, E. 1994. *Features and Fluents. The Representation of Knowledge about Dynamical Systems. Volume I.* Oxford University Press.
11. Sandewall, E. 1996. Towards the Validation of High-level Action Descriptions from their Low-level Definitions. *Linköping electronic articles in Computer and Information science*, Vol. 1 (1996): 4.
12. Sandewall, E. 1997. Logic-based Modelling of Goal-Directed Behaviour. *Linköping electronic articles in Computer and Information science*, Vol. 2 (1997): 19.
13. Steels, L. 1990. Exploiting Analogical Representations. In Maes, P. (ed) *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, 71 – 88, Mass.: MIT/Elsevier.