Modelling Situations in Intelligent Agents [*]

John Thangarajah
RMIT University
Melbourne, Australia
johthan@cs.rmit.edu.au

Lin Padgham
RMIT University
Melbourne, Australia
linpa@cs.rmit.edu.au

Sebastian Sardina
RMIT University
Melbourne, Australia
ssardina@cs.rmit.edu.au

General Terms

Design, Theory

Keywords

Agent and multi-agent architectures; Frameworks, infrastructures and environments for agent systems

1. INTRODUCTION AND MOTIVATION

BDI agent systems and languages such as PRS, JAM, JACK, 3APL, and AGENTSPEAK have been widely used in developing robust and flexible applications in dynamic domains. However, one criticism of these systems is that the modelling of how agent reasoning progresses is too reliant on the rather low level notion of individual *events*. In our own work in a number of application areas, we have consistently noticed a need for a more abstract concept, which we call a *situation*. Recognition by the agent that it is in a particular situation may affect the goals that it has, may place overarching constraints on how it operates, or may influence the way that it chooses to achieve its goals.

Work on "situation awareness" [1] also supports the position that aggregation of lower level perceptions into meaningful chunks allowing for understanding is a crucial part of intelligent agent systems. Work on "situation spaces" [2] collects together information about the environment into a set of meaningful high-level abstractions that provide the context for the agent's cognitive activity (in this case planning). Situation awareness has also more recently been proposed as a means of interpreting pro-active behaviour in agents [3].

While situations can in a sense be captured simply as a collection of beliefs, there is an advantage in introducing them as a conceptual entity which can be modelled declaratively.[1] Firstly, this allows for developing a semantics that directly captures the meaning of situations within the agent architecture (for example, we shall require that the reasoning about situations takes priority over all other agent reasoning.) Secondly, it allows for domain experts to more readily specify the characteristics of the situations particular to the domain. Thirdly, it can allow for general purpose infrastructure which supports the recognition of situations and the appropriate adjustment of agent attitudes.

As an initial step in capturing and reasoning about situations, we provide an operational semantics describing the agent's reasoning when it recognises that a particular situation exists, or ceases to exist. We do this within the context of the previously presented BDI agent language CAN (**C**onceptual **A**gent **N**otation) [4], which is similar in style to AGENTSPEAK, 3APL, and other BDI languages. The distinguishing feature of CAN is its goal construct and its account of failure handling typically found in implemented BDI systems, which provides goal persistence.

Although our initial work is assuming situations will be recognised by individual agents, the work is generalisable to multi-agent systems also.

2. SITUATION RULES

We define the rules for managing situations as an addition to the CAN language defined in [4]. As previously indicated, situations are (conceptually) at a coarser level of granularity than individual events, and the agent's reaction to a situation may involve changing its mental attitudes and focus of attention, rather than directly producing behaviour. Consequently, we introduce the notion of a *meta* program/plan (MP) whose objective is not to produce behaviour (directly), but to change such things as attitude, focus, understanding, and priorities of an agent. It is possible to conceive a range of things a meta plan could do, such as change the agent's goals; drop active intentions; change the plan library; start protecting certain conditions; or engage in specialised reasoning such as diagnosis or prediction. For the purpose of the current work, though, we shall restrict meta plans to add goals and to set and release system constraints.

In the CAN language, an *agent configuration* is a tuple $\langle \mathcal{N}, \Pi, \mathcal{B}, \mathcal{G}, \Gamma \rangle$ where $\mathcal{N}$ is the agent name, Π is a plan library, $\mathcal{B}$ is a belief base, $\mathcal{G}$ is a goal base, and Γ is the

[*]We thank Bao Quoc Vo, within the RMIT agents group, for contributions to this work. We also gratefully acknowledge the support of the Australian Research Council, The Bureau of Meteorology and AOS (especially Ralph Rönnquist), under Grant LP0347025 and Grant LP0453486.

[1]We note that plan types in the BDI plan library can also be modelled simply as beliefs, However, it is well accepted that there is both a computational and semantic advantage in modelling plans as special kinds of beliefs. We claim the same is true of situations.

set of current intentions. The belief base of an agent often contains ground belief atoms in the form of first-order relations. However, CAN makes no commitments about the form of the belief base; sophisticated logics and reasoning about action formalisms can be used to model more expressive belief bases. We simply require that well-defined operations exist to check whether a condition follows from a belief set ($\mathcal{B} \models \phi$), to add a belief b to a belief set ($\mathcal{B} \cup \{b\}$), and to delete a belief b from a belief set ($\mathcal{B} \setminus \{b\}$). Observe that, in order to capture interesting situations, a temporal formalism for beliefs is likely to be desirable. We will discuss this below when talking about the implementation.

Next, we introduce a *constraint* construct, $\mathsf{Constraint}(\alpha, P)$, with the intended meaning that no action is to be performed if it would make the agent not believe α. If at some point α does not hold, initially or due to some factor beyond the agent's control (e.g., other agent's actions), then plan P ought to be executed. Program P is intended to encode a recovery procedure to re-establish α. We denote the set of constraints in the system as $\mathcal{C}$.

In the CAN language, actions are primitives that always succeed. In order to reason about constraints we require this notion of actions to be extended, as we want to restrict an agent from performing any action that violates any system constraint. To that end, we assume that agents are equipped with a STRIPS-like action description library Λ containing rules of the form: $act : \psi_{act} \leftarrow \ {}^{-}_{act};\ {}^{+}_{act}$, one for each action type in the domain, where ψ_{act} is the pre-condition of the action, and ${}^{+}_{act}$ and ${}^{-}_{act}$ are the add and delete lists of atoms, respectively. Figure 1 shows the derivation rules for actions. Note that rule *act* ensures that the agent only performs an action if the system constraints are not violated. If the action does violate a system constraint, then the action does not fail but is not allowed to execute. This means that the action in question would be re-considered later on. Hence, actions only *fail* if the pre-conditions are not met.

So, we next define a meta plan MP as follows:

$$\begin{aligned} MP ::= &+\mathsf{Goal}(\phi_s, P, \phi_f) \mid +\mathsf{Constraint}(\alpha, P) \mid \\ &-\mathsf{Constraint}(\alpha, P) \mid MP_1; MP_2 \mid true \end{aligned}$$

where P is a CAN plan-body and MP_1, MP_2 are meta plans.

A situation library Π_{Sit} contains all the situations the agent knows of. Concretely, Π_{Sit} includes situation tuples of the form $\langle N, \phi_a, MP_a, \phi_e, MP_e \rangle$, where N is the name of the situation, ϕ_a and ϕ_e are entry and exit conditions respectively, and MP_a and MP_e are the meta plans that the agent executes on situation entry and exit.

We extend the notion of an agent configuration to be $\langle \mathcal{N}, \Pi, \Lambda, \Pi_{Sit}, \mathcal{B}, \mathcal{G}, \mathcal{C}, \mathcal{S}, \Gamma \rangle$, where Π_{Sit} is a situation library, $\mathcal{C}$ is the set of current system constraints, and $\mathcal{S}$ is the set of situations that the agent currently believes to be active. Each entry in $\mathcal{S}$ is the name of the situation and the object bindings that enabled the entry condition of the situation to be true. Figure 2 depicts the derivation rules for capturing when a particular situation becomes active and when a situation ends (Sit_{active} and Sit_{end}), as well as for handling meta-plans. Rule Sit_{active} states that for any situation $\langle N, \phi_a, MP_a, \phi_e, MP_e \rangle$, the situation becomes active when: (a) ϕ_a is true with the bindings ; and (b) the situation in question is not already active (that is, N is not already in $\mathcal{S}$.[2] When a new situation becomes active, then the agent: (a) creates an entry N in the active situations set; and (b) executes the steps of the meta plan MP_a with the corresponding bindings to ensure that the plan acts upon the same object instances that formed the situation.

[2]This is necessary to avoid the agent repeatedly reacting to the very same situation.

The Sit_{end} rule states that for any situation that is currently active, if the exit condition is true, then the agent: (a) removes the relevant entry from the active situations set; and (b) executes the exit meta plan with the object bindings that enabled the exit condition to hold. We shall require that the Sit_{active} and Sit_{end} rules take precedence over all other rules of the agent, i.e., they must be applied whenever applicable in every BDI execution cycle.

3. REPRESENTING AND MONITORING

We have indicated that situations will be characterised by some sort of belief-set formulae indicating entry to and exit from a situation, along with meta plans which will be executed on entry and exit. Situations often include a recognition of trends and changes that have occurred over a period of time such as "the thunderstorm's intensity has been increasing steadily in the last 30 minutes" or "the thunderstorm has been moving towards the fire." In order to support this kind of reasoning, we choose to maintain a temporal database. The challenge then is to know what should be stored in this temporal database in order to support the relevant belief-set queries, and when we should query this database.

Noting that the temporal queries we encounter generally have to do with *appearances* of objects over time, we choose to explicitly identify those objects which are involved in temporal relations in the recognition of the entry or exit condition of some situation. We refer to this set of objects as the core of a situation as it contains the core objects which require monitoring over time for situation recognition. This identification enables us to maintain historical information regarding only the relevant object types, and to query the temporal database at selected points to recognise entry to and exit from a situation.

The entry condition for the situation is then specified as a temporal entry formula (TEF) and a current entry formula (CEF). If the TEF is true with some bindings, the CEF will then also be checked with these same bindings. These are conceptually part of the same formula, but are separated out to facilitate using the temporal database only where necessary, if that is desired. The exit condition for a situation is specified in a similar manner. Along with the entry and exit conditions, we specify the meta-plans that ought to be executed when these conditions hold.

We also introduce the notion of a set of active situations, which are those situation instances where the entry condition has been true, and the exit condition has not yet become true. Situation instances are added to and removed from this set dynamically. This set is important to ensure that the agent does not repeatedly act upon the same situation instance, and to keep track of which situation instances the end condition should be monitored for.

During execution we need to ensure that relevant temporal (and other) information is stored, while also ensuring that there is not a blow-out in space consumption. We then need to use this information to recognise when a situation starts and ends. We chose to use a temporal database

$$\frac{a:\quad \leftarrow\quad {}^{+};\quad {}^{-} \in \Lambda \quad a\theta = act \quad \mathcal{B} \models \quad \theta \quad \forall \mathsf{Constraint}(\alpha, P) \in \mathcal{C} : (\mathcal{B} \setminus \quad {}^{-}\theta) \cup \quad {}^{+}\theta \models \alpha}{\langle \mathcal{B}, \mathcal{G}, \mathcal{C}, act \rangle \longrightarrow \langle (\mathcal{B} \setminus \quad {}^{-}\theta) \cup \quad {}^{+}\theta, \mathcal{G}, \mathcal{C}, true \rangle} \; act$$

$$\frac{a:\quad \leftarrow\quad {}^{+};\quad {}^{-} \in \Lambda \quad a\theta = act \quad \mathcal{B} \not\models \quad \theta}{\langle \mathcal{B}, \mathcal{G}, \mathcal{C}, act \rangle \longrightarrow \langle \mathcal{B}, \mathcal{G}, \mathcal{C}, fail \rangle} \; act_f$$

Figure 1: Derivation rules for actions over basic configurations.

$$\frac{\langle N, \phi_a, MP_a, \phi_e, MP_e \rangle \in \Pi_{Sit} \quad \mathcal{B} \models \phi_a\theta \quad (N, \theta) \notin \mathcal{S} \quad \langle \mathcal{B}, \mathcal{G}, \mathcal{C}, \Gamma, MP_a\theta \rangle \overset{*}{\Longrightarrow}_{Sit} \langle \mathcal{B}', \mathcal{G}', \mathcal{C}', \Gamma', true \rangle}{\langle \Pi_{Sit}, \mathcal{B}, \mathcal{G}, \mathcal{C}, \mathcal{S}, \Gamma \rangle \longrightarrow \langle \Pi_{Sit}, \mathcal{B}', \mathcal{G}', \mathcal{C}', \mathcal{S} \cup \{N, \theta\}, \Gamma' \rangle} \; Sit_{active}$$

$$\frac{\langle N, \phi_a, MP_a, \phi_e, MP_e \rangle \in \Pi_{Sit} \quad \mathcal{B} \models \phi_e\theta\theta' \quad (N, \theta) \in \mathcal{S} \quad \langle \mathcal{B}, \mathcal{G}, \Gamma, \mathcal{C}, MP_e\theta\theta' \rangle \overset{*}{\Longrightarrow}_{Sit} \langle \mathcal{B}', \mathcal{G}', \Gamma', \mathcal{C}', true \rangle}{\langle \Pi_{Sit}, \mathcal{B}, \mathcal{G}, \mathcal{C}, \mathcal{S}, \Gamma \rangle \longrightarrow \langle \Pi_{Sit}, \mathcal{B}', \mathcal{G}', \mathcal{C}', \mathcal{S} \setminus \{N, \theta\}, \Gamma' \rangle} \; Sit_{end}$$

$$\frac{\langle \mathcal{B}, \mathcal{G}, \mathcal{C}, \Gamma, MP_1 \rangle \Longrightarrow_{Sit} \langle \mathcal{B}', \mathcal{G}', \mathcal{C}', \Gamma', MP_1' \rangle}{\langle \mathcal{B}, \mathcal{G}, \mathcal{C}, \Gamma, (MP_1;MP_2) \rangle \Longrightarrow_{Sit} \langle \mathcal{B}'', \mathcal{G}'', \mathcal{C}'', \Gamma'', (MP_1';MP_2) \rangle} \; ;_1 \qquad \frac{}{\langle \mathcal{B}, \mathcal{G}, \mathcal{C}, \Gamma, (true;MP_2) \rangle \Longrightarrow_{Sit} \langle \mathcal{B}, \mathcal{G}, \mathcal{C}, \Gamma, MP_2 \rangle} \; ;_2$$

$$\frac{}{\langle \mathcal{B}, \mathcal{G}, \mathcal{C}, \Gamma, +\mathsf{Constraint}(\alpha, P) \rangle \Longrightarrow_{Sit} \langle \mathcal{B}, \mathcal{G}, \mathcal{C} \cup \{\mathsf{Constraint}(\alpha, P)\}, \Gamma, true \rangle} \; +\mathsf{Constraint}$$

$$\frac{}{\langle \mathcal{B}, \mathcal{G}, \mathcal{C}, \Gamma, -\mathsf{Constraint}(\alpha, P) \rangle \Longrightarrow_{Sit} \langle \mathcal{B}, \mathcal{G}, \mathcal{C} \setminus \{\mathsf{Constraint}(\alpha, P)\}, \Gamma, true \rangle} \; -\mathsf{Constraint}$$

$$\frac{}{\langle \mathcal{B}, \mathcal{G}, \mathcal{C}, \Gamma, +\mathsf{Goal}(\phi_s, P, \phi_f) \rangle \Longrightarrow_{Sit} \langle \mathcal{B}, \mathcal{G}, \mathcal{C}, \Gamma \cup \{\mathsf{Goal}(\phi_s, P, \phi_f)\}, true \rangle} \; +\mathsf{Goal}$$

Figure 2: Rules for situations and meta plans. $\Longrightarrow_{Sit}$ is a transition relation between situation configurations of the form $\langle \mathcal{B}, \mathcal{G}, \mathcal{C}, \Gamma, MP \rangle$ ($\overset{*}{\Longrightarrow}_{Sit}$ stands for the usual reflexive transitive closure of $\Longrightarrow_{Sit}$.)

to store object information, as they provide an infrastructure for storing and retrieving objects with temporal data. The particular temporal database system that we are currently investigating is RapidBase[3] [5]. There are a number of reasons for this choice, some of which we outline below. The RapidBase Query Language (RQL) is based on SQL, which allows programmers with sufficient SQL skills to be able to write RQL queries. RapidBase allows tables to be defined with history columns, such that only histories of those columns will be maintained. This helps in avoiding redundant data and keeping the size of the database to a minimum.

We use RapidBase to store and track objects over time. For efficiency reasons, we only record objects whose types are part of the *core* of some situation known to the agent. Objects are stored by creating a table for each object type and specifying the attributes that require histories to be stored, as history columns.

When specifying a known situation, entry and exit conditions are expressed as RQL temporal queries. If a query returns an empty result set, then the condition in question is false; otherwise, when the result set is not empty, the answer set will contain the bindings for the objects for which the corresponding condition is true. These bindings are necessary for the entry condition, because, as mentioned previously: (a) the same bindings must be used to check the exit condition for that situation instance; and (b) the meta-plan that executes on activation should also use the same bindings as they should reason about the same object instances that formed the situation. The manner in which the object bindings are obtained is another useful and convenient feature of using a temporal database.

[3]`http://www.vtt.fi/tte/projects/rapid/`

4. DISCUSSION AND CONCLUSION

We have introduced a notion of a situation as a special type of entity of which agents should be aware. We have added some rules to a previously developed agent specification language which provide for some initial aspects of responding to situations. We have also presented a representation for situations at the application programming level using a temporal database to capture the important temporal comparisons that commonly occur. We have currently only addressed a small number of things to be done upon recognition of a situation. This is an area where future work is required.

5. REFERENCES

[1] M. R. Endsley. Toward a theory of situation awareness in dynamic systems. *Human Factors*, 37(1):32–64, 1995.

[2] C. F. Schmidt, J. L. Goodson, S. C. Marsella, and J. L. Bresina. Reactive planning using a "situation space". In *Proceedings of the Annual AI Systems in Government Conference*, pages 50–55, March 1989.

[3] Raymond So and Liz Sonenberg. Situation awareness in intelligent agents: Foundations for a theory of proactive agent behavior. In *IEEE/WIC/ACM IAT*, 2004.

[4] Michael Winikoff, Lin Padgham, James Harland, and John Thangarajah. Declarative & procedural goals in intelligent agent systems. In *Proceedings of KR*, 2002.

[5] A. Wolski, J. Kuha, T. Luukkanen, and A. Pesonen. Design of RapidBase - an Active Measurement Database System. In *Proceedings of IDEAS 2000*, pages 75–82. IEEE Computer Society Press, 2000.