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

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.

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 (Conceptual Agent Notation) [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 (N, Π, B, G, Γ) where N is the agent name, Π is a plan library, B is a belief base, G is a goal base, and Γ is the

---

1. 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 Rönquist), 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 \((B \models \phi)\), to add a belief \(b\) to a belief set \((B \cup \{b\})\), and to delete a belief \(b\) from a belief set \((B \setminus \{b\})\). 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, \(\text{Constraint}(\alpha, P)\), with the intended meaning that no action is to be performed if it would make the agent not believe \(\alpha\). If at some point \(\alpha\) 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 \(\alpha\). 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. A containing rules of the form: \(\text{act} : \psi_{\text{act}} \leftarrow \text{act} \_ \text{act}, \) one for each action type in the domain, where \(\psi_{\text{act}}\) is the pre-condition of the action, and \(\text{act} \_ \text{act} \) and \(\text{act} \_ \text{act}\) are the add and delete lists of atoms, respectively. Figure 1 shows the derivation rules for actions. Note that rule \(\text{act}\) 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:

\[
MP ::= +\text{Goal}(\phi_s, P, \phi_f) | +\text{Constraint}(\alpha, P) | -\text{Constraint}(\alpha, P) | MP_1 ; MP_2 | \text{true}
\]

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

A situation library \(\Pi_{Sit}\) contains all the situations the agents know of. Concretely, \(\Pi_{Sit}\) includes situation tuples of the form \(\langle N, \phi_s, MP_a, \phi_o, MP_p\rangle\), where \(N\) is the name of the situation, \(\phi_s\) and \(\phi_o\) are entry and exit conditions respectively, and \(MP_a\) and \(MP_p\) are the meta plans that the agent executes on situation entry and exit.

We extend the notion of an agent configuration to be \(\langle N, \Pi, \Lambda, \Pi_{Sit}, \mathcal{B}, \mathcal{G}, \mathcal{C}, \mathcal{S}, \Gamma\rangle\), where \(\Pi_{Sit}\) 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 \(\langle N, \phi_s, MP_a, \phi_o, MP_p\rangle\), the situation becomes active when: (a) \(\phi_s\) is true with the bindings \(\cdot\); and (b) the situation in question is not already active (that is, \(N\) is not already in \(\mathcal{S}\)).

When a new situation becomes active, then the agent:

\footnote{This is necessary to avoid the agent repeatedly reacting to the very same situation.}

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
Figure 1: Derivation rules for actions over basic configurations.

\[ a : \leftarrow \gamma ; \exists \alpha \uparrow \alpha = a \theta \mapsto 0 \iff \forall \text{Constraint}(\alpha, P) \in \text{C} : (B \setminus \theta) \cup \alpha \mapsto \alpha \mapsto \text{act} \]

\[ \langle B, G, C, \text{act} \rangle \rightarrow \langle (B \setminus \theta) \cup \alpha, G, C, \text{true} \rangle \]

\[ \langle B, G, C, \text{act} \rangle \rightarrow \langle (B, G, C, \text{fail} \rangle \]

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

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


\(^{3}\text{http://www.vtt.fi/tte/projects/rapid/}^{3}\)