

User Defined Monitoring Strategies for BDI Agent Programs *

Timothy William Cleaver
Griffith University
Southport, Australia
Tim.Cleaver@student.
griffith.edu.au

Abdul Sattar
Griffith University
Brisbane, Australia
a.sattar@griffith.edu.au

Raihana Ferdous
Griffith University
Southport, Australia
Raihana.Ferdous@student.
griffith.edu.au

ABSTRACT

An agent needs to balance its time between reflecting on its past behaviors, the execution of its current behaviors and the recognition of opportunities for future action. The optimal strategy to achieve this balance is heavily dependent on the environment in which the agent is situated. In this paper we extend a BDI agent programming language called AgentSpeak(L) in two ways. Firstly we refine the use of execution monitoring to enable agents to reflect on their past endeavors. Secondly, we extend the semantics and syntax of the language to allow for user defined monitoring strategies. This allows agent designers to utilize execution monitoring in a way that balances the focus of an agents behavior in a way that is appropriate for its application domain.

Keywords: Agent and Multi-Agent Architectures, Agent Programming Languages

1. INTRODUCTION

For an intelligent agent, behaving in complex and real-time environments is a balancing act. There are three primary tasks that must be managed in order to achieve this balance. An agent must be able to recognize opportunities for future conduct. Simultaneously, the agent must devote resources, including its cognitive abilities, towards the execution of its current behaviors. All the while the agent must be watchful to collate its perceptions with the outcomes of its previous behaviors so as to maintain an accurate view of its progress towards its goals. This must all be done while remaining responsive to real-time constraints on its behavior.

This paper proposes the use of execution monitoring to allow BDI systems to correlate their perceptions with their previous behavior. More specifically we argue that providing the ability for agent developers to integrate user defined monitoring strategies is the only way to do this efficiently given the variety of domains for which BDI systems are applicable. AgentSpeak(L) was chosen as the prototypical BDI system for reasons outlined in Section 3 and it is the framework into which we have integrated execution monitoring.

*This work is supported by the SIT-CRC

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

AAMAS'06 May 8–12 2006, Hakodate, Hokkaido, Japan.
Copyright 2006 ACM 1-59593-303-4/06/0005 ...\$5.00.

toring. However, prior to providing an overview of AgentSpeak(L) and our extensions (Section 4) we describe the execution monitoring problem for BDI systems.

2. THE MONITORING PROBLEM

Monitoring is the mental act of validating expectations through the inspection of ones world model. There are three main things that a BDI system should be monitoring for: success, validity and failure. In dynamic and complex environments, monitoring for one cannot be reduced to monitoring for another[5]. Ideally an agent should continuously monitor the environment to determine the outcomes of its behaviors. However, in resource-bounded practical agents the monitoring process has a cost. Therefore an agent needs to schedule the times at which it monitors. This way an agent makes a trade-off between its introspection capabilities and the time constraints imposed upon it by the environment in which it is embedded. Thus, an agent needs to minimize the cost of monitoring. Be it the cost resulting from not monitoring when monitoring would have been beneficial or the cost of the monitoring process itself when the monitoring process has no effect on the agent's behavior.

3. WHAT IS AGENTSPEAK(L)?

AgentSpeak(L)[9] is an abstract architecture and programming language that aims to embody the BDI philosophy[4] and logic[10] in a computationally attractive way. It has a number of semantics ranging from the Z specification language [7] to structural operational semantics [8, 3, 1] and a sub-set of the language has been demonstrated, through model-checking, to satisfy a number of specifications expressed in BDI logic formula[2]. Due to its simplicity and elegance, AgentSpeak(L) is easy to extend and is designed in such a way that customizing the architecture and language to suit the application domain is straight forward.

4. MONITORING IN AGENTSPEAK(L)

Recently, the AgentSpeak(L) language and architecture was extended to facilitate the specification of and reasoning with success and failure conditions of plan execution[5, 6]. However, one significant drawback of the approach proposed was that the architecture utilized a naïve approach to monitoring execution. Each success and failure condition of each intended means in each intention stack was checked once each reasoning cycle. This results in significant proportions of the agents cognitive faculties being devoted to the monitoring process at the expense of its other activities. In order to balance the monitoring process with action execution, opportunity recognition and belief revision an agent needs to be able to make decisions as to when to monitor for the success and failure of its

plans. We believe that the optimal strategy is dependent on both the application domain and the deployment environment. To cater for this we incorporate monitoring strategies by way of an additional user function in the architecture of the agent. We build upon the structural operational semantics and syntax developed in [1] and extended in [5, 6]. The primary contribution of this paper required no modification to the extended syntax of the language. However it was necessary to add new definitions, auxiliary functions and rules to the semantics of the interpreter.

4.1 Definitions

We extend the definition of an agent to include two new user defined functions: F_M and F_{ConRes} . F_M encapsulates the agents monitoring strategy. Given that both implemented BDI systems and BDI logic[10] represent time discreetly, this function takes the current state of the agent, a mental attitude to monitor and maps them to true or false ($M : S \times At \mapsto [\top, \perp]$). True indicates the agent needs to check the condition this reasoning cycle and false indicates the agent need not. $F_{ConRes} : \xi \times B \mapsto [\top, \perp, \neg]$ defines the agents conflict resolution strategy. F_{ConRes} is necessary because the agent may come to believe evidence for both the success and failure of an effect of a plans execution. It is used to alleviate the ambiguity of whether to conclude success or failure in these cases. $F_{ConRes} : \xi \times B \mapsto [\top, \perp, \neg]$ maps an effect (ξ) of a plan and the current beliefs (B) of the agent to one of three values: $\top$ if the agent should conclude true given this ambiguity, $\perp$ if the agent should conclude false under uncertainty and $\neg$ if the agent can conclude nothing this cycle and should continue monitoring this effect.

One additional temporary data store is also required: η . During each reasoning cycle, this set is generated based on the user function F_M to include all the effects that the agent needs to check for either success or failure. Being temporary (the contents do not persist across reasoning cycles), this data store is referred to in the semantics as T_η .

4.2 Auxiliary Functions

Here we define the filtering function (**FilterInt** : $C_I \mapsto \{\pi\}$). It returns the sub-set of the current set of intentions (C_I) such that each included intention has elements remaining in the body. This function is used to prevent intentions that have completed from executing but have effects outstanding from being candidates for processing on this reasoning cycle.

$$\mathbf{FilterInt}(C_I) = \{\pi | \pi = \pi' [l \rightarrow te : c \leftarrow \{i\}h.e] \wedge h \neq \emptyset\}$$

This function (**GenEff** : $T \times C \times AG \mapsto \{\langle \pi, \varrho, \xi \rangle\}$) generates the set of effects that should be monitored this reasoning cycle. This is achieved by iterating over each intention in the set of intentions ($\forall \pi$), each plan on each intention stack ($\forall \varrho$) and each effect within each plan ($\forall \xi$). This combination, in addition to the current state of the agent (T, C), is passed onto the user defined effect monitoring strategy function ($F_M(T, C, \pi, \varrho, \xi)$). This function defines whether this effect should be checked in this cognitive cycle. The return value of this function is the set of tuples $\langle \pi, \varrho, \xi \rangle$ for which the ($F_M(T, C, \pi, \varrho, \xi)$) function returns true ($\top$).

$$\mathbf{GenEff}(T, C, AG) = \left\{ \begin{array}{l} \langle \pi, \varrho, \xi \rangle \quad \forall \pi \varrho \xi \text{ s.t. } \pi \in C_I \wedge \varrho \in \pi \wedge \\ l \rightarrow te : c \leftarrow \{i\}h.e = \varrho \wedge \\ \xi \in e \wedge AG_{F_M(T, C, \pi, \varrho, \xi)} = \top \end{array} \right\}$$

Because we regenerate the set of effects to monitor each reasoning cycle, we need to remove those effects that have sufficient evidence for the agent to conclude their success from their encompassing plan. Otherwise, the agent will continue to monitor for

these effects and the plan will never be considered complete. We do this using the update effects function (**UpdEff** : $\pi \times \varrho \times \xi \mapsto \pi'$). This function maps an intention, plan and effect into a new intention. Because an agent can monitor for and consequently conclude the success of an effect in any level of an intention stack, this function retrieves the input plan ($l \rightarrow te : c \leftarrow \{i\}h.e$) from the intention (π)¹, removes the input effect (ξ) from the input plan ($l \rightarrow te : c \leftarrow \{i\}h.e$) and reconstructs the intention stack (π') to be returned.

$$\mathbf{UpdEff}(\pi, l \rightarrow te : c \leftarrow \{i\}h.e, \xi) = \pi'$$

where: $\pi'' [l \rightarrow te : c \leftarrow \{i\}h.e \setminus \xi] \pi'''$ s.t. $\pi'' [l \rightarrow te : c \leftarrow \{i\}h.e] \pi''' = \pi$

4.3 Rules

The first rule that we must alter from the original is that of **SelInt**. We do so in order to integrate the intention filtering function (as described above) into the intention selection process. The first rule is applicable in the case where there are intentions for which further processing is required. The second handles the case where there are not.

$$\mathbf{SelInt} \frac{\mathbf{FilterInt}(C_I) \neq \emptyset \quad F_I(\mathbf{FilterInt}(C_I)) = T_I}{\langle AG, C, T, \mathbf{SelInt} \rangle \longrightarrow \langle AG, C, T', \mathbf{ExecInt} \rangle}$$

$$\mathbf{SelInt} \frac{\mathbf{FilterInt}(C_I) = \emptyset}{\langle AG, C, T, \mathbf{SelInt} \rangle \longrightarrow \langle AG, C, T, \mathbf{GenEff} \rangle}$$

We do not repeat the rules for executing the top most plan of the selected intention as they remain identical to those in [3]. The next process is that of generating the set of effects to check. We employ two rules to do this. The first handles the case where the set of intentions is empty. If the set of intentions is empty then there can be no effects to be monitoring for and we skip the procedure and transition to clearing the intentions.

$$\mathbf{GenEff} \frac{C_I = \emptyset}{\langle AG, C, T, \mathbf{GenEff} \rangle \longrightarrow \langle AG, C, T, \mathbf{ClrInt} \rangle}$$

However, should there be elements in the set of intentions the second rule passes the current set of intentions as a parameter to the effect generation function and stores the result in the set of effects to be checked (T_η).

$$\mathbf{GenEff} \frac{C_I \neq \emptyset \quad \mathbf{GenEff}(T, C, AG) = T_\eta}{\langle AG, C, T, \mathbf{GenEff} \rangle \longrightarrow \langle AG, C, T', \mathbf{UpdEff} \rangle}$$

Four rules are required to process the current effects. The first three are repeatedly applied until there are no elements left in the effect set. The first is applicable when there are outstanding effects to be checked ($T_\eta \neq \emptyset$) and the selected effect is deemed successful ($\top = F_{ConRes}(\xi, B)$) by the conflict resolution strategy (F_{ConRes}). It removes the effect from the set of effects awaiting monitoring and from the plan from which it originated. It also updates the intention using the (**UpdEff**) function.

$$\mathbf{UpdEff} \frac{T_\eta \neq \emptyset \quad \langle \pi, \varrho, \xi \rangle \in T_\eta \wedge \top = F_{ConRes}(\xi, B)}{\langle AG, C, T, \mathbf{UpdEff} \rangle \longrightarrow \langle AG, C', T', \mathbf{UpdEff} \rangle}$$

where:

$$T'_\eta = T_\eta \setminus \langle \pi, \varrho, \xi \rangle$$

$$C'_I = (C_I \setminus \pi) \cup \mathbf{UpdEff}(\pi, \varrho, \xi)$$

¹producing $\pi'' [l \rightarrow te : c \leftarrow \{i\}h.e'] \pi'''$ where π'' and π''' represent the elements of the intention both below and above the required plan respectively

The second rule is also applicable when there are outstanding effects to be checked, but requires the conflict resolution strategy to judge the selected effect to have failed. This rule creates the appropriate failure event and adds it to the event set. Then the intention is removed from the set of intentions to prevent its further processing until the failure is handled.

$$\text{UpdEff} \frac{T_\eta \neq \emptyset \quad \langle \pi, \varrho, \xi \rangle \in T_\eta \wedge \perp = F_{ConRes}(\xi, B)}{\langle AG, C, T, \text{UpdEff} \rangle \longrightarrow \langle AG, C', T', \text{UpdEff} \rangle}$$

$$\text{where: } C'_E = \begin{cases} C_E \cup \{(+g, \pi)\} & \text{if } te = -g \text{ (where } g \text{ is a goal)} \\ C_E \cup \{(-g, \pi)\} & \text{if } te = +g \text{ (where } g \text{ is a goal)} \end{cases}$$

$$C'_I = C_I \setminus \pi$$

$$T'_\eta = T_\eta \setminus \langle \pi, \varrho, \xi \rangle$$

The third rule handles the case where there are effects to audit, one of these is selected, yet the conflict resolution strategy can not conclude that the effect has been satisfied or invalidated. All that must be done in this case is to remove the effect from the effect set so that it will not be tested again.

$$\text{UpdEff} \frac{T_\eta \neq \emptyset \quad \langle \pi, \varrho, \xi \rangle \in T_\eta \wedge \neg = F_{ConRes}(\xi, B)}{\langle AG, C, T, \text{UpdEff} \rangle \longrightarrow \langle AG, C', T', \text{UpdEff} \rangle}$$

$$\text{where } T'_\eta = T_\eta \setminus \langle \pi, \varrho, \xi \rangle$$

Finally, the last (**UpdEff**) rule handles the case where there are no elements in the set of effects to review.

$$\text{UpdEff} \frac{T_\eta = \emptyset}{\langle AG, C, T, \text{UpdEff} \rangle \longrightarrow \langle AG, C, T, \text{ClrInt} \rangle}$$

There is only one primary modification required to the original (**ClrInt**) function and that is to retain plans that have outstanding effects. This prevents the continued processing of these intentions until all the effects have been realized or demonstrated to have failed.

$$\text{ClrInt} \frac{C_I \neq \emptyset}{\langle AG, C, T, \text{ClrInt} \rangle \longrightarrow \langle AG, C', T', \text{SelEv} \rangle}$$

$$\text{where: } C'_I = \begin{cases} C_I \setminus T_i & \text{if } T_i = [l \rightarrow te : c \leftarrow \{i\}h.e] \\ & \wedge i = h = e = \emptyset \\ (C_I \setminus T_i) \cup \{T'_i\} & \text{if } T_i = \pi[l \rightarrow te : c \leftarrow \{i\}!at; h.e] \\ & [l' \rightarrow te' : c' \leftarrow \{i'\}h'.e'] \wedge \\ & i' = h' = e' = \emptyset \\ C_I & \text{otherwise} \end{cases}$$

$$T = \emptyset$$

5. CONCLUSION

In conclusion, through our studies of existing BDI systems we discovered that these systems do not have strong capabilities for monitoring the outcomes of their behaviors. Particularly, existing systems have no methodology for utilizing user defined monitoring strategies that facilitate effective balance of the time an agent spends monitoring and doing its other activities. We chose as our prototypical BDI implementation, AgentSpeak(L), and made extensions to its semantics in an attempt to address this issue. Due to the sensitivity of the performance of an agent to the environment in which it is situated, utilizing user defined strategies that are tuned to the particular application domain is essential. We believe that agents utilizing these extensions will perform better due to their ability to better balance the time spent correlating their current beliefs with their past behaviors, the time spent recognizing opportunities and executing current behaviors.

6. REFERENCES

- [1] R. Bordini, M. Dastani, J. Dix, and A. Seghrouhni. *Multi-Agent Programming: Languages, Platforms and Applications*. Springer, 2005.
- [2] R. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model checking AgentSpeak. In *Proceedings of the second International Joint Conference on Autonomous Agents and Multi-agent Systems*, pages 409–416, 2003.
- [3] R. Bordini and A. Moreira. Proving BDI properties of agent-oriented programming languages: The asymmetry thesis principles in agentspeak(l). *Annals of Mathematics and Artificial Intelligence*, 42:197–226, 2004.
- [4] M. Bratman. *Intention, Plans and Practical Reason*. Harvard University Press, 1987.
- [5] T. Cleaver, A. Sattar, and K. Wang. Reasoning about success and failure in intentional agents. In *Proceedings of the 2005 Pacific Rim International Workshop on Multi-Agents*, 2005.
- [6] T. Cleaver, A. Sattar, and K. Wang. Reasoning with plan outcomes in intentional agents. In *Proceedings of the eighteenth Australian Joint Conference on Artificial Intelligence (to appear)*, 2005.
- [7] M. d’Inverno and M. Luck. Engineering AgentSpeak(L): A Formal Computational Model. *Journal of Logic and Computation*, 8(3):1–27, 1998.
- [8] A. Moreira, R. Vieira, and R. Bordini. Extending the Operational Semantics of a BDI Agent-Oriented Programming Language for Introducing Speech-Act based Communication. In J. Leite and A. Omicini and L. Sterling and P. Torroni, editor, *Proceedings of the first International Workshop on Declarative Agent Languages and Technologies*, volume 2990 of *Lecture Notes in Computer Science*, pages 135–154. Springer, 2003.
- [9] A. Rao. Agentspeak(L): BDI Agents Speak out in a Logical Computable Language. In W. van de Velde and J. Perrame, editors, *Proceedings of the seventh European Workshop on Modeling Autonomous Agents in a Multi-Agent World*, volume 1038 of *Lecture Notes in Artificial Intelligence*, pages 42–55. Springer-Verlag, 1996.
- [10] A. Rao and M. Georgeff. Decision procedures for BDI logic. *Journal of Logic and Computation*, 8(3), 1998.