Belief/Goal Sharing BDI Modules

(Extended Abstract)

Michal Cap[1,2]*, Mehdi Dastani[1] and Maaike Harbers[1]

[1]Intelligent Systems Group, Faculty of Science, Utrecht University, Utrecht, Netherlands
{mehdi,maaike}@cs.uu.nl
[2]ATG, Dept. of Cybernetics, FEE, Czech Technical University, Prague, Czech Republic
cap@agents.felk.cvut.cz

ABSTRACT

This paper proposes a modularisation framework for BDI based agent programming languages developed from a software engineering perspective. Like other proposals, BDI modules are seen as encapsulations of cognitive components. However, unlike other approaches, modules are here instantiated and manipulated in a similar fashion as objects in object orientation. In particular, an agent's mental state is formed dynamically by instantiating and activating BDI modules. The agent deliberates on its active module instances, which interact by sharing their beliefs and goals.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Intelligent agents, languages and structures*

General Terms

Theory, Design, Languages

Keywords

Agent programming languages, BDI, Modularity

1. INTRODUCTION

The agent oriented programming paradigm promotes a societal view of computation, where solutions are achieved by cooperation of autonomous entities - agents. This paper focuses on a family of agent programming languages based on the Belief-Desire-Intention (BDI) theory [3]. BDI languages (e.g. [2]) offer constructs inspired by mental notions such as beliefs, goals and plans to implement agent behaviour. As in other programming paradigms, the ability to decompose BDI programs to separate, to some extent independent modules, is crucial for the development of complex software systems. Yet, a widely accepted concept of modularisation for BDI programming languages is still missing.

We propose a modularisation framework for logic-based BDI languages to overcome some of the limitations of existing frameworks and unify commonly accepted characteristics of various existing approaches [4, 7–9] into one single framework. The proposed framework extends earlier work by Dastani et al [5, 6].

2. BDI MODULARISATION

Our proposed modularisation framework for the BDI programming languages has the following characteristics. 1) A module is an encapsulation of beliefs, goals, plans and reasoning rules that together specify a functionality, a capability, a role, or a behaviour. 2) An agent's mental state is modelled as a tree of module instances, in which a link is created when one module instance activates another. Using this mechanism, a set of dependent module instances can be deactivated and reactivated by means of a single action. 3) An agent's module instances are executed in parallel. This allows the agent to play several roles or use several capabilities at the same time. 4) Module instances can be created and released, and added to or removed from an agent's mental state at run-time. This can be used, e.g., to dynamically enact and deact roles. 5) Inactive and active module instances are distinguished. An inactive module instance is generally used as a named container for beliefs and goals, while an active module instance is typically used for encapsulation of behavioural rules (specifying plans to achieve goals and respond to events). 6) Each module instance is associated with an interface determining its interaction with other module instances, i.e. the beliefs and goals that are shared with other module instances. This way, a module's public interface is separated from its private internals. 7) An agent's module instances can be clustered into separate belief/goal sharing scopes. Modules in different scopes do not interact which allows an agent to maintain mutually inconsistent belief bases, e.g. to model different possible worlds or profiles of other agents.

From a methodological point of view, we can identify the following characteristics. 1) The framework is easy to grasp for programmers acquainted with object orientation because module instances are manipulated similarly to objects. 2) Programmers have explicit control over the life cycle of a module, i.e. they can indicate when to create/in-

*The research was done at Utrecht University. The first author is now affiliated with CTU Prague and supported by the Grant Agency of the Czech Technical University in Prague, grant no. SGS10/189/OHK3/2T/13 and by the Czech Ministry of Education grant no. MSM6840770038.

Cite as: Belief/Goal Sharing BDI Modules (Extended Abstract), Michal Cap, Mehdi Dastani and Maaike Harbers, *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, Tumer, Yolum, Sonenberg and Stone (eds.), May, 2–6, 2011, Taipei, Taiwan, pp. 1201-1202.

stantiate modules, how to operate on them, and when to release them. 3) The module interface can be used to determine the intended use of a module. By convention, the use of particular interface should be documented by a semi-formal comment (similar to JavaDoc comments) above the respective interface entry. 4) Active module instances interact by sharing some of their beliefs and goals which promotes loose coupling. A module instance can easily be replaced (even at runtime) as long as the new module uses the same beliefs and goals for interaction with the agent's other modules.

3. BELIEF/GOAL SHARING

The mechanism of belief/goal sharing, which realizes the run-time interaction between active module instances, is a distinguishing feature of our approach. Each module instance has an interface which is defined as a set of interface entries. An interface entry is an atomic formula used as a template that matches concrete beliefs and goals. All beliefs and goals of a module instance matched by its interface are exported and become global beliefs and goals of a sharing scope, and vice versa, all global beliefs and goals of a sharing scope matched by the module's interface are imported and treated identically to its own beliefs and goals.

A module interface serves several functions. First, it specifies the language that is to be used to interact with the module. All beliefs and goals interfaced by the module instance will be expressed in the module interface language. Second, a module interface defines which of its local beliefs and goals are interfaced and will thus be constitute beliefs and goals of its sharing scope. Third, a module interface defines which of the global beliefs and goals will be accessible for the module instance. And last, a module interface may be used to limit the visibility of the internals of a module instance. Any belief or goal that cannot be expressed in terms of the module interface language stays private and cannot be accessed from outside the module instance.

We introduce a simple example to demonstrate one of the typical interaction patterns exploiting the belief/goal sharing mechanism — the delegation of a goal pursuit. Suppose we are specifying a worker agent who operates in a grid-like environment. The agent consists of the main `worker` module instance and a `moving` module instance providing the agent a capability to move in the environment. The agent's module tree is depicted in Figure 1.

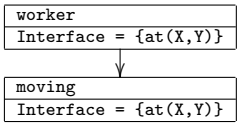

Figure 1: Modules of the Worker Agent

A goal pursuit is delegated when a module instance is incapable to achieve that goal itself, but another module instance in the same sharing scope is capable to achieve it. The first module instance can monitor the pursuit of the goal by a query on the corresponding belief. In our example, the `working` module instance may desire to be at position (5,7) , i.e. it adopts the goal `at(5,7)`, although it has no actual means to achieve the goal itself. However, since the atom `at(X,Y)` is declared as an interface entry in the `worker` module specification, the goal `at(5,7)` will be exported and becomes a global goal of the agent. The `moving` module specification also declares the atom `at(X,Y)` in its interface, and therefore imports the global goal. Subsequently, it generates a plan to perform actions in the external environment towards the achievement of the goal. Eventually, the `moving` module instance will have sensed that the agent is at the target position and updates its belief base with a new position belief `at(5,7)`. Using the belief sharing mechanism, the belief gets propagated back to the `worker` module. Furthermore, due to the rationality principle[1] the goal `at(5,7)` is automatically dropped.

4. CONCLUSION

We have designed a belief/goal sharing modularisation framework suitable for BDI-based agent programming languages with declarative goals. It shares some of its characteristics with the other approaches and adds several novel features. The concept of belief/goal sharing was outlined using a simple example. We have used the open source codes of 2APL to incorporate the proposed constructs into this programming language and implemented an interpreter able to execute such modular programs [1].

5. REFERENCES

[1] `http://apapl.sourceforge.net/`.

[2] R. Bordini, M. Dastani, J. Dix, and A. E. F. Seghrouchni. *Multi-Agent Programming: Languages, Platforms and Applications*. International book series on Multiagent Systems, Artificial Societies, and Simulated Organizations. Springer, 2005.

[3] M. E. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, Cambridge, MA, 1987.

[4] L. Braubach, A. Pokahr, and W. Lamersdorf. Extending the capability concept for flexible BDI agent modularization. In *Proceedings of PROMAS 2005 Workshop*. Springer Verlag, 2006.

[5] M. Dastani. 2APL: a practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, 16(3):214–248, 2008.

[6] M. Dastani, C. P. Mol, and B. R. Steunebrink. Modularity in BDI-based agent programming languages. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, 2009.

[7] K. Hindriks. Modules as policy-based intentions: Modular agent programming in GOAL. In *Proceedings of PROMAS '07 Workshop, number 4908 in LNAI*. Springer, 2008.

[8] P. Novák and J. Dix. Modular BDI architecture. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. ACM, 2006.

[9] M. B. van Riemsdijk, M. Dastani, J.-J. C. Meyer, and F. S. de Boer. Goal-oriented modularity in agent programming. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. ACM, 2006.

[1]The rationality principle states that an agent should not desire a worlstate which is believed to hold. Some BDI languages (e.g. 2APL) enforce this principle by automatically dropping goals that are believed to hold.