Enabling BDI Group Plans with Coordination Middleware: Semantics and Implementation

JAAMAS Track

Stephen Cranefield
University of Otago
Dunedin, New Zealand
stophen.cranefield@otago.ac.nz

ABSTRACT

This extended abstract summarises an article with the same title published in the journal Autonomous Agents and Multi-Agent Systems [6].

KEYWORDS

BDI agents; Group plans and goals; Coordination middleware

ACM Reference Format:
Stephen Cranefield. 2022. Enabling BDI Group Plans with Coordination Middleware: Semantics and Implementation: JAAMAS Track. In *Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), Online, May 9–13, 2022*, IFAAMAS, 3 pages.

1 INTRODUCTION

Belief-Desire-Intention (BDI) agent programming provides a powerful and popular model for developing software that is goal-oriented, adaptive to its circumstances, and equipped with pre-existing domain knowledge in the form of plans. However, the BDI architecture is a model for a *single* practical reasoning agent, and does not address the question of how to create a group of agents that need to coordinate their actions while achieving some common goals.

Many theories, models and systems have addressed this question, as outlined in the full paper. Prominent approaches include: (i) systems based on the *joint intention theory* [11] and related work, which provide rich logical accounts of the interplay between agents' beliefs, goals and intentions during the execution of group tasks, but require agent reasoning capabilities that are, in general, beyond the in-built capabilities of BDI agent platforms; and (ii) practical agent programming tools that extend BDI agent platforms with support for team activities, such as JACK Teams [1] and JaCaMo [2], but which lack formal semantics that relate them to the underlying BDI model. Alternatively, (iii) the agent programmer(s) can manage coordination by explicitly sending messages (or invoking coordination middleware) to coordinate agents' plan executions and to propagate knowledge of their success and/or failure—thus increasing the code complexity for all team members, and compromising correctness, scalability and robustness.

This work aims to provide an alternative approach for programming agent teams that provides a straightforward extension of the BDI agent programming model, encapsulates the coordination of individual agent behaviour, and has formal semantics that extend the standard BDI semantics.

Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), P. Faliszewski, V. Mascardi, C. Pelachaud, M.E. Taylor (eds.), May 9–13, 2022, Online.

2 GROUP GOALS AND PLANS

The approach allows the *declarative* specification of the coordination required between agents in a team, using a new programming abstraction: the *group goal*. A group goal specifies a set of subgoals, one for each member of the group to perform, that must all be achieved for the overall group goal to succeed. A group goal becomes active when one or more agents begin working towards it, and can then eventually fail or succeed. As group goals are only satisfied when all agents' subgoals are completed, the successful completion of each group goal is a point of synchronisation between agents. In other words, an agent that quickly completes its subgoal for one group goal cannot begin work on the next group goal until all other agents complete their subgoals. This is in line with the idea that a group plan is a mechanism for specifying the coordinated action of a group of agents. To avoid agents waiting indefinitely for other agents to complete their subgoals, we allow a group goal to fail due to a *join timeout* or a *completion timeout*. A join timeout occurs when some agents do not begin working on their subgoals within a certain time interval after the earliest start of any of the other agents' subgoals. A completion timeout occurs when the result of some subgoals are still unknown within a certain interval since the last agent began work on its subgoal.

From the programmer's point of view, a group goal is called directly, via a small supporting library of generic plans that handle the required coordination with other agents *implicitly* using robust industry-grade coordination middleware (Apache Camel [9] and ZooKeeper [8]). These generic plans ensure the group goal will succeed only when all agents begin and complete their plans in a timely manner. This makes the programmer's task simpler as the plans that he or she writes will be less complex and more maintainable.

We conceptualise a group plan as one that defines a collective view of coordinated action amongst a group of agents, and which can be followed in a synchronised way by all agents. We rely on group goals to provide the coordination between agents, and define a group plan as one that includes group goals. Group plans may be complex and include any control structures provided by the BDI programming language used. A group plan is supported by a set of individual plans for each agent to achieve its local subgoals. These individual plans could be known by all agents, or they could be kept private, as the application and group requires. Group goal timeout values are specified as part of a group plan.

Group plans can be *dynamically* acquired—they do not need to be loaded in advance into organisational middleware as, for example, in JaCaMo [2]. A single top-level group plan, containing group

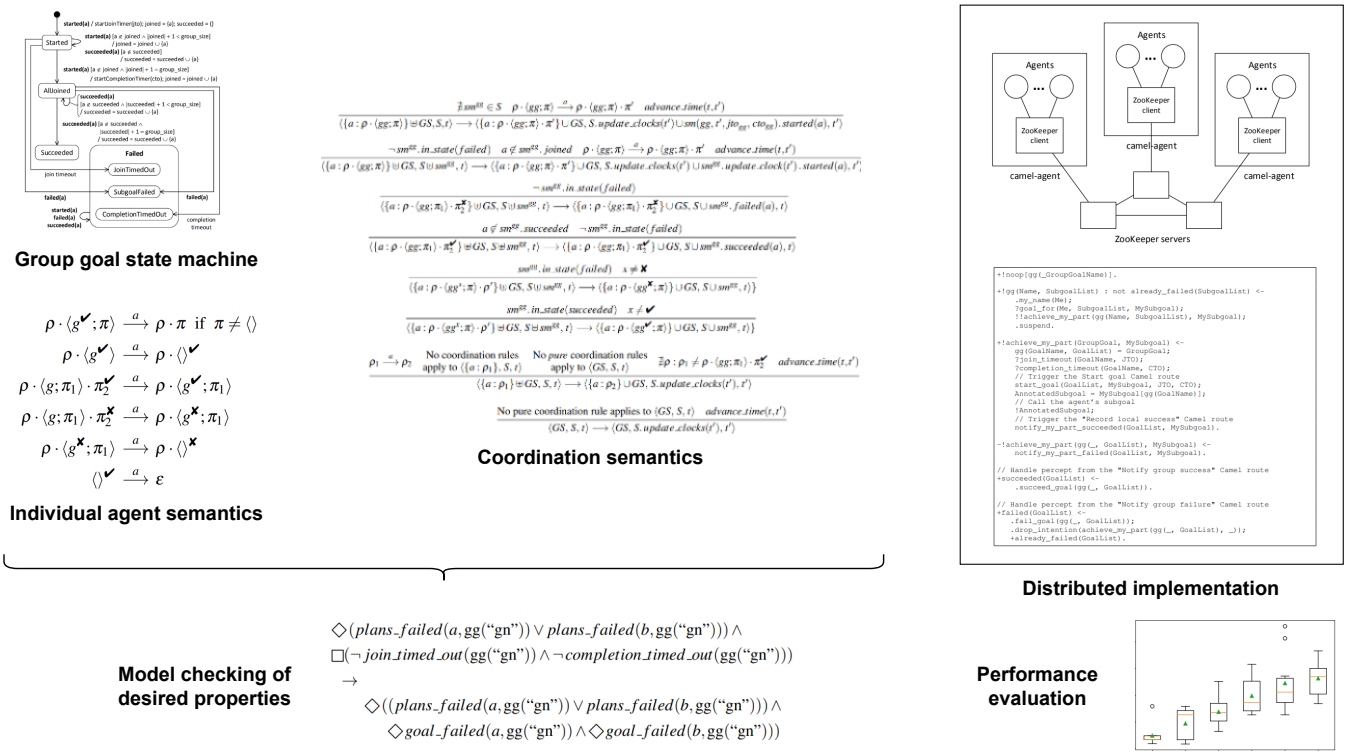

Figure 1: Overview of our methodology

goals, can be shared amongst all group members to serve as the blueprint for their collective action (supported by their own local plans for their individual subgoals).

3 METHODOLOGY

Figure 1 illustrates our methodology. The semantics of group goals are defined in terms of a state machine (top left) defining how individual agents starting, succeeding and failing their local subgoals affect the state of a group goal. The semantics for individual agent goal executions are defined as transition rules on intention stacks (middle left), and a set of coordination rules (upper middle) define how these local semantics are modified for a group goal, depending on its state. The aim is to distinguish the situations in which individual agents can perform their BDI reasoning independently of the other agents, and those in which their computations must update the state machine for a group goal or wait for a group to reach a particular state.

A set of desired properties for group goal execution were verified using LTL model checking, given a direct translation of the semantics into the Maude equational logic programming language [5] (one property is shown in the lower middle of the figure). This approach allowed the semantics to be refined iteratively, with a number of subtle bugs corrected, until all six properties were verified. The properties state that timeouts should happen in the correct circumstances and have the effect of failing a group goal, and that when no timeouts occur, all agents should eventually detect either the success of a group goal if all agents successfully complete their local plans, or the failure of a group goal if any local plan failure occurs.

The right hand side of the figure illustrates our implementation of group goals for Jason [3] agents, based on a set of domain-independent plans for interacting with group goals. These plans maintain a connection to the industry-grade Apache ZooKeeper coordination middleware, via Apache Camel [9] service integration routes and the camel-agent Camel component [7]. A performance evaluation showed that the implementation is scalable.

4 FUTURE WORK

In future work, it would be useful to support other types of group goals, e.g. disjunctions of local subgoals or relaxed conjunctions where at least m out of n agents must participate to achieve success. This would also require different timeout models to be designed. A current limitation of our approach is that free variables in group goals and local subgoals are not supported: an instantiation of a variable in a local subgoal will not cause the instantiation of the same variable in the group goal to be shared across all group members. Communication of variable bindings between agents can be implemented using similar middleware techniques to those we use to implement group goals [10], and the same approach could also be used to provide a notion of shared group beliefs. On the implementation side, a comparison could be made between our implementation approach using ZooKeeper and alternative middleware tools that can maintain an eventually consistent shared state, such as Cassandra [4].

REFERENCES

[1] Agent Oriented Software Pty Ltd. 2005. JACK Intelligent Agents Teams Manual, Release 5.5. https://aosgrp.com/media/documentation/jack/JACK_Teams_Manual.pdf.

[2] Olivier Boissier, Rafael H. Bordini, Jomi F. Hübner, Alessandro Ricci, and Andrea Santi. 2013. Multi-Agent Oriented Programming with JaCaMo. *Science of Computer Programming* 78, 6 (2013), 747–761.

[3] Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. 2007. *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley.

[4] Jeff Carpenter and Eben Hewitt. 2022. *Cassandra: The Definitive Guide* (revised third ed.). O'Reilly.

[5] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. 2003. The Maude 2.0 System. In *Rewriting Techniques and Applications*. Lecture Notes in Computer Science, Vol. 2706. Springer, 76–87.

[6] Stephen Cranefield. 2021. Enabling BDI Group Plans with Coordination Middleware: Semantics and Implementation. *Autonomous Agents and Multi-Agent Systems* 35, 2 (2021), 45. https://doi.org/10.1007/s10458-021-09525-7

[7] Stephen Cranefield and Surangika Ranathunga. 2013. Embedding Agents in Business Processes Using Enterprise Integration Patterns. In *Engineering Multi-Agent Systems*. Lecture Notes in Computer Science, Vol. 8245. Springer, 97–116.

[8] Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed. 2010. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In *Proceedings of the USENIX Annual Technical Conference*. USENIX Association.

[9] Claus Ibsen and Jonathan Anstey. 2018. *Camel in Action* (second ed.). Manning.

[10] Edmund Soon Lee Lam, Iliano Cervesato, and Nabeeha Fatima. 2015. Comingle: Distributed Logic Programming for Decentralized Mobile Ensembles. In *Coordination Models and Languages – 17th IFIP WG 6.1 International Conference, COORDINATION 2015*. Lecture Notes in Computer Science, Vol. 9037. Springer, 51–66.

[11] Hector J. Levesque, Philip R. Cohen, and José H. T. Nunes. 1990. On Acting Together. In *Proceedings of the 8th National Conference on Artificial Intelligence*. AAAI Press / The MIT Press, 94–99.