# Logic and Agent Programming Languages

Natasha Alechina

School of Computer Science
University of Nottingham
nza@cs.nott.ac.uk

**Abstract.** Agent programming languages based on the Belief, Desire and Intentions (BDI) framework present some interesting challenges for logicians. While BDI logics have been studied extensively, problems related to belief update and analysis of plans in BDI agent programming languages have received less attention.

## 1 Introduction

This paper describes work in progress and proposes some possible research questions. It first introduces main ideas of agent programming languages and then describes some interesting open problems in agent programming language design that relate to logic, database theory and reasoning about actions. The paper is targeted mainly at logicians and not at agent programming languages researchers, and presents agent programming languages in a somewhat simplified way, abstracting away from many subtle differences between them. The aim of the paper is to encourage more logicians to investigate open problems in this area. Most of the questions discussed in this paper arose in discussions with agent programming languages researchers: Brian Logan, Koen Hindriks, Mehdi Dastani and Rafael Bordini.

## 2 Agent Programming Languages

There are many definitions of 'agent' in the literature [1]. Key ideas include:

**autonomy:** an agent operates without the direct intervention of humans or other agents
**situatedness:** an agent interacts with its environment (which may contain other agents)
**reactivity:** an agent responds in a timely fashion to changes in its environment
**proactivity:** an agent exhibits goal-directed behaviour

Arguably the dominant agent programming paradigm is the Belief, Desire and Intentions (BDI) model [2]. The BDI model is based on the work of Michael Bratman [3], and views an agent as a computational system whose behaviour can be usefully characterised in terms of propositional attitudes such as beliefs and goals, and which is programmed in an agent programming language that makes

explicit use of propositional attitudes. *BDI agent programming languages* are designed to facilitate the implementation of BDI agents. A BDI agent programming language has programming constructs corresponding to beliefs, desires and intentions. An agent *architecture* or *interpreter* enforces relationships between the agent's beliefs, desires and intentions, and causes the agent to choose actions to achieve its goals based on its beliefs. One of the first BDI agent programming languages was the Procedural Reasoning System (PRS) [4]. Example applications of PRS include space shuttle fault diagnosis, controlling a mobile robot, air traffic control, business process control etc. PRS was very influential, with many derivatives (e.g., PRS-CL, PRS-Lite, dMARS), and modern BDI agent programming languages such as AgentSpeak(L) [5], CAN [6], SPARK [7], Jason [8], 2APL [9], and Goal [10] are based on similar ideas.

The basic idea of a BDI agent programming language is that the agent's state contains beliefs (about the agent's environment), goals (states the agent desires to bring about), and plans (sequences of actions the agent intends to carry out). *Beliefs* are typically represented as facts and Horn clause rules. For example, an agent may have the following beliefs about the world:

$$Location(home)$$

$$Seaside(x) \leftarrow NextTo(sea, x)$$

*Goals* or desires are usually conjunctions of literals describing desired states of affairs the agent would like to achieve. For example:

$$Location(x) \wedge Seaside(x)$$

is a goal to be at the seaside ($x$ implicitly existentially quantified; any seaside would do).

Finally, agents have a set of *plans* (intentions) they have adopted and are in the process of executing. In most BDI languages plans look like imperative programs (sequential composition, if tests and while loops) which may contain variables in actions and tests. For example:

$$\text{if } (Location(x) \wedge Seaside(y) \wedge CheapFlight(x, y)) \; buyTicket(x, y); go(x, y)$$

This means: find some substitution for $x$ and $y$ which satisfies

$$Location(x) \wedge Seaside(y) \wedge CheapFlight(x, y)$$

and then execute a sequence of actions

$$buyTicket(x, y); go(x, y)$$

for those values of $x$ and $y$. The sets of beliefs, goals and plans are referred to as the agent's belief base, goal base and plan base.

The agent operates in a cycle:

**belief update** The agent checks what is happening in the environment and updates its belief base accordingly.

**intention adoption** The agent decides, using its current beliefs and goals, which new plans to adopt (if any), and adds them to its plan base.

**intention execution** The agent decides which actions (forming part of current intentions) to execute, and executes them.

## 3 Briefly: BDI logics

The development of first agent programming languages was accompanied by the development of BDI logics, which formalise logical relationships between beliefs, desires and intentions. See, for example, [11, 12], and subsequent work, for example, [13, 14].

The logics study questions such as relationships between beliefs and goals, when a rational agent should adopt and drop intentions, should logical consequences of intentions be intended, etc. These are deep questions, and the rational properties of the relationship between, for example, beliefs and goals proposed by different authors are sometimes completely the opposite of each other. For example, Cohen and Levesque [12] gave a standard possible world semantics for $BEL$ and $GOAL$ modalities, requiring that goal-accessible (desirable) worlds are a subset of belief-accessible worlds. This makes sense because of all the worlds the agent considers possible, only some are desirable, and it does not make sense to have desirable states which are not considered possible. This property corresponds to the axiom:

$$BEL\,\phi \rightarrow GOAL\,\phi$$

On the other hand, Rao and Georgeff [11] have a more complex semantics where possible worlds are trees (branching histories) and for each belief-accessible history, there is a goal-accessible sub-history inside it, intutitively representing those courses of events which the agent finds desirable. Achievement goals are expressed by existential temporal formulas (there is a future state satisfying the goal), which are preserved under extensions. This means that for an existential temporal formula $\phi$, the following axiom holds:

$$GOAL\,\phi \rightarrow BEL\,\phi$$

Finally, in modern agent programming languages, achievement goals correspond to state properties (describing a desirable state). Since it does not make sense for the agent to intend what it already believes is achieved, the following axiom holds:

$$GOAL\,\phi \rightarrow \neg BEL\,\phi$$

There are clearly problems associated with defining beliefs and goals as standard modalities, similar to logical omniscience in standard epistemic logics, and it is possible to define BDI logics which do not have this problem: see, for example, [15].

There is a lot of interesting work on BDI logics, but this paper focusses on other aspects of agent programming languages where logical reasoning is involved: the belief update phase and the intention adoption phase of the BDI agent execution cycle.

# 4 Belief update

In a typical modern BDI agent programming language, the set of beliefs is essentially a deductive database. It consists of ground atoms (facts, EDB) and Horn clause rules, which define a different set of predicates (IDB). However, there are extensions of programming languages such as Jason which include ontologies and other sources of definitions in the belief base [16], and people would really like agents to be able to do some temporal reasoning [17].

## 4.1 Standard belief update

Let us use the term 'standard belief update' to refer to belief update for the case when the agent's belief base is a deductive database, and an update is represented as two sets of ground atoms $A^+$ and $A^-$ (an add list and a delete list). $A^+$ are the facts which have become true, and $A^-$ are the facts which have become false. The idea of course is that $A^+ \cap A^- = \emptyset$. Then the result of updating a belief base $S$ with $(A^+, A^-)$ is $S' = (S \cup A^+) \setminus A^-$. There is nothing else to do to ensure the belief base is consistent, since the sets of EDB and IDB predicates are disjoint and hence 'false' facts can not be derived from the update.

Usually, at this stage, those elements of the goal base which have become true (derivable from the belief base) are removed from the goal base. (The reason for this is that in case of declarative achievement goals, there is no point trying to achieve them if they are already true.)

## 4.2 Standard belief update in the presence of query caching

The intention adoption and intention execution stages may involve evaluating queries against the agent's belief base (and possibly goal base). If the results of queries are cached (remembered for further use), the belief update problem becomes less trivial (since it involves updating the cache).

First we discuss the queries which need to be evaluated and why it is a good idea to cache them.

The intention adoption phase requires checking which plans are applicable given the agent's beliefs and possibly given its goals. A typical approach is to have 'plan adoption rules' of the form

$$\phi \leftarrow \psi \mid \pi$$

which mean 'if query $\phi$ succeeds against the goal base, and, extending the same substitution, query $\psi$ succeeds against the belief base, then adopt the plan $\pi$ with the resulting substitution'. Queries are usually built from literals (where $\neg$ is interpreted as negation as failure, and any variables in its scope should also appear in the query in a positive literal) using disjunctions and conjunctions.

For example:

$$on(x, y) \leftarrow block(x) \wedge block(y) \wedge \neg(x = y) \wedge clear(x) \wedge clear(y) \mid stack(x, y)$$

The intention execution phase may require evaluating belief tests, as in the previous example

$$\text{if } (Location(x) \wedge Seaside(y) \wedge CheapFlight(x,y)) \; buyTicket(x,y); go(x,y)$$

However, studying existing programs in various agent programming languages suggests that the agents repeatedly evaluate the same queries in the same cycle and across different cycles [18], so it does make sense to cache complex derived formulas.

Assume that in addition to the belief base $S$, we also have a set of cached query results $C$. Each element of $C$ is a pair $(\phi, \theta)$ where $\phi$ is a query and $\theta$ is a substitution which currently makes $\phi$ true. Depending on the agent programming language, a query may return a set of answers (substitutions) or a single substitution.

In the presence of caching, the belief update problem becomes more involved than in the standard case, since we need to decide how to update $C$ given $S$ and $(A^+, A^-)$. (The result of updating $S$ with $(A^+, A^-)$ is defined as before: $S' = (S \cup A^+) \setminus A^-$.) In fact we have two problems:

**invalidation of cached beliefs** Given a change of belief base from $S$ to $S'$, which of the cached beliefs should be removed from $C$?

**maintaining a complete cache** Given a change of belief base from $S$ to $S'$, are there are any new elements which need to be added to $C$? (In other words, when does it make sense to re-evaluate a query to get new answers?) The second problem occurs when we want to cache *all* answers to a query.

**Invalidation of cached beliefs** The problem of invalidating cached beliefs can be solved using well known AI reason maintenance techniques [19]. We can keep track of which facts in $S$ were used in deriving a particular element of $C$, and store them together with the element of $C$ whose truth depends on them. In line with AI reason-maintenance terminology, let us call a set of literals used to derive a certain element $(\phi, \theta)$ of $C$ an *environment* for $(\phi, \theta)$, and the set of all environments for $(\phi, \theta)$ a *label* for $(\phi, \theta)$. There may be several environments in one label, for example if $S$ contains

$$P(x) \leftarrow Q_1(x), R(x)$$

$$P(x) \leftarrow Q_2(x), R(x)$$

and $Q_1(a)$, $Q_2(a)$ and $R(a)$ are all in $S$, then $(P(x), x/a)$ has a label with two environments, $\{Q_1(a), R(a)\}$ and $\{Q_2(a), R(a)\}$. Let us say that an environment is 'destroyed' if one of its elements is no longer in $S$. For example, if $Q_1(a)$ is no longer in $S$, then $\{Q_1(a), R(a)\}$ is destroyed. Destroyed environments are removed from the label. If some element of $C$ has an empty label, it should be removed from $C$, because all old ways of deriving it have disappeared from $S$.

This works in a straightforward way if $\phi$ is an atomic query, and it is easy to see how to generalise this to conjunctions and disjunctions of atomic queries.

Negations are more of a problem, since we have negation as failure which is not stored as an explicit element of $S$. If we cache queries with negations, or if rules can have negative literals in the body, we need to store negative literals in the environments. For example, let $S$ be

$$P(x) \leftarrow Q_1(x), R(x)$$
$$P(x) \leftarrow Q_2(x), R(x)$$
$$Q_1(a), Q_2(a), R(a), R(b)$$

and let $\phi$ be $R(x) \wedge \neg P(x)$. Given $S$, we have

$$(R(x) \wedge \neg P(x), x/b) \in C$$

and the only environment in its label is $\{R(b), \neg Q_1(b), \neg Q_2(b)\}$. An environment containing both positive and negative literals is destroyed by an update $(A^+, A^-)$ if either one of its positive literals is in $A^-$, or for some negative literal $\neg L$, $L \in A^+$. Clearly, this cache update operation is much less computationally expensive than a call to the inference engine. Building and maintaining labels of cache elements in the case of only positive environments not more expensive that a call to an inference agent to derive them in the first place. In the case of environments with negative elements, efficient environment computation appears to be an open problem (it is obviously do-able, but can it be done as efficiently as computing positive environments?).

**Maintaining a complete cache** The second problem (maintaining a complete cache) makes sense if we want the cache to always contain all answers to a query.

The question then is, given a set of cached queries and an update, which new substitutions should be added? Clearly, this question can be solved in a straightforward way by re-deriving all the queries, but we are looking for an efficient way of at least detecting when a new call to an inference engine is required (and ideally, just reducing this to a pattern matching problem on $S, A^+$ and $A^-$). In other words, we need to maintain a relationship between literals and complex queries such that if new literals of a certain form are added or removed, then a new substitution should be added to a query. It is likely that to solve this problem efficiently one would need to be able to compute something like a RETE or TREAT network [20, 21] but not for Horn clause rules but for arbitrary boolean combinations of literals and with negative literals as possible tokens.

**Open problems** Here are some open problems related to cache update:

**efficient standard update in the presence of a cache:** how to compute and maintain environments with negative literals, how to efficiently maintain a complete cache when the agent's beliefs correspond to a deductive database

**extensions to the standard case:** how to update the cache if an update can contain disjunctions of literals; for which fragments of description logics and temporal logic cache update problems can be solved efficiently

## 5 Maintaining the plan base

In general the agent's plan base will contain several plans. These plans may be executed simultaneously, e.g., in a round robin fashion or an arbitrarily interleaved order. For example, the agent may be stacking blocks on several tables at the same time, using several robotic arms or moving from table to table. In some cases, making progress on several tasks at the same time is a good idea, while in other cases plans may be interfere in unfortunate ways. The latter problem is intuitively clear, but how do we define a 'rational plan base'? There are several aspects to this notion, some to do with whether it makes sense to have several plans for the same goal (usually not) and others to do with how much work an agent can rationally commit to (considered at the end of this section). First, let us clarify the notion of 'interfering plans'.

One approach, see for example [22], considers it a bad idea to adopt plans for conflicting (logically inconsistent) goals, for example a goal to be in Amsterdam and a goal to be in Paris. A plan which involves achieving one of them will prevent achievement of another. Whether goals conflict is reasonably straightforward to check.

Another meaning of interfering plans [23–25] is that executing steps of one plan is undoing or making impossible executing steps of another plan, even if the goals of the two plans are consistent. For example, it is consistent to have two goals $Clean(room1)$ and $Clean(room2)$ (where $room1$ and $room2$ are different rooms in different directions from the agent), but most obvious plans for doing this would involve going to room 1 and going in the opposite direction to room 2.

Consider a blocks world environment [26]. Plans which involve manipulating blocks on the same table may interfere in both senses. Two goals may be inconsistent, for example $On(a, b)$ and $On(b, a)$.

An example of a situation where the goals are consistent but the plans may interfere is as follows. Suppose we have blocks $a$, $b$, $c$, $d$ and $e$, and $a$ and $e$ are red and the rest are green. Suppose the first goal is to have a red block on top of two green blocks where the bottom block is on the table, and the second goal is to have a green block on top of a red block which is on the table. Both goals are achievable separately and even together, for example we can have

$$On(a, b) \wedge On(b, c) \wedge OnTable(c)$$

satisfying the first goal, and

$$On(d, e) \wedge OnTable(e)$$

satisfying the second goal. However suppose the following plans are adopted to achieve them (for example in the state where the agent believes that all blocks are on the table and clear):

$$\text{if } (OnTable(x) \wedge Green(x) \wedge Green(y))\ Stack(y, x); \text{if } (Red(z))Stack(z, y)$$

for the first goal and

$$\text{if } (OnTable(x) \wedge Red(x) \wedge Green(y))Stack(y, x)$$

If both plans generate the same substitution for picking a red block ($a$) then one of them will become unexecutable, given the obvious pre- and postconditions of the *Stack* action ($Stack(x, y)$ is executable if $x$ and $y$ do not have anything on top of them, and result in $On(x, y)$ which means that $x$ is on top of $y$).

Before formulating the problems related to checking for interference of plans, we need some definitions.

A *schedule* (of a set of actions) is an assignment of actions to processors (e.g. robotic arms) together with a linear order of each set of actions assigned to the same processor. For example, a schedule for two robotic arms could be:

$$arm1: \ Stack(b, c) \prec Stack(a, b)$$

$$arm2: \ Stack(b, a)$$

where $\prec$ is the linear order (temporal precedence) relation.

Two actions *interfere* (or one of them clobbers another) if executing one of them makes execution of another impossible; in other words, the effect of the first action makes precondition of the second action false. In the schedule above, $Stack(b, a)$ interferes with $Stack(a, b)$ since the precondition of $Stack(a, b)$ requires that $a$ has nothing on top of it, and the effect of $Stack(b, a)$ is that it does.

**Open problems** There are a number of problems relating to the interference of plans, including:

**plan interference** given a set of plans $P$ and a set of pre- and postconditions of actions in those plans, do these plans interfere?

**interference-free schedule** given a set of plans $P$ and a set of pre- and postconditions of actions in those plans, return a schedule (if one exists) of actions of plans in $P$ where action do not interfere

**schedule interference** given a set of plans $P$, a set of pre- and postconditions of actions in those plans, and a schedule of actions in the plans, check whether actions in the schedule interfere

Clearly, these problems relate to planning and scheduling and can be solved using existing methods; however it would be good to find more efficient solutions which are tailored to plans in agent programming languages. One approach would be to investigate special scheduling algorithms which make use of logical checks before the plans are adopted to make scheduling an easier problem. It may also be possible to come up with a weaker notion of an acceptable ('almost free from interference') set of plans, for which the scheduling problem can be solved in polynomial time.

There are other aspects to deciding whether a set of plans the agent is committed to is 'rational', apart from interference of plans. For example, the goals

the agent is trying to achieve may have deadlines. Given the time required to execute its plans, the agent may not be able to achieve all its goals, and will waste time and energy executing plans which are certain not to achieve their goal in time. Related problems have been investigated in [29–31]. In [31], we proposed an efficient scheduling algorithm which did not return the optimal schedule (which is computationally expensive) but a schedule satisfying reasonable 'rationality criteria'. The same algorithm was used in a plan adoption algorithm for an agent which decides whether to commit to a set of obligations (which also have deadlines) in [32]. However, in this area much work remains to be done, trying to find a balance between computationally efficient and ideally rational criteria for plan schedules.

## References


1. Wooldridge, M., Jennings, N.R.: Intelligent agents: Theory and practice. Knowledge Engineering Review **10**(2) (Jun 1995) 115–152
2. Georgeff, M.P., Pell, B., Pollack, M.E., Tambe, M., Wooldridge, M.: The Belief-Desire-Intention model of agency. In Müller, J.P., Singh, M.P., Rao, A.S., eds.: Intelligent Agents V, Agent Theories, Architectures, and Languages, 5th International Workshop, (ATAL'98), Paris, France, July 4-7, 1998, Proceedings. Volume 1555 of Lecture Notes in Computer Science., Springer (1999) 1–10
3. Bratman, M.: Intention, Plans, and Practical Reason. Harvard University Press (1987)
4. Georgeff, M.P., Lansky, A.L.: Reactive reasoning and planning. In: Proceedings of the Sixth National Conference on Artificial Intelligence, AAAI-87. (1987) 677–682
5. Rao, A.S.: Agentspeak(l): Bdi agents speak out in a logical computable language. In de Velde, W.V., Perram, J.W., eds.: Agents Breaking Away, 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Eindhoven, The Netherlands, January 22-25, 1996, Proceedings. Volume 1038 of Lecture Notes in Computer Science., Springer (1996) 42–55
6. Winikoff, M., Padgham, L., Harland, J., Thangarajah, J.: Declarative & procedural goals in intelligent agent systems. In Fensel, D., Giunchiglia, F., McGuinness, D.L., Williams, M.A., eds.: Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR-02), Toulouse, France, Morgan Kaufmann (April 2002) 470–481
7. Morley, D., Myers, K.: The SPARK agent framework. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04), Washington, DC, USA, IEEE Computer Society (2004) 714–721
8. Bordini, R.H., Hubner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak using Jason. Wiley (2007)
9. Dastani, M.: 2APL: a practical agent programming language. Journal of Autonomous Agents and Multi-Agent Systems **16**(3) (2008) 214–248
10. Hindriks, K.V.: Programming rational agents in goal. In El Fallah Seghrouchni, A., Dix, J., Dastani, M., Bordini, R.H., eds.: Multi-Agent Programming: Languages, Tools and Applications. Springer US (2009) 119–157
11. Rao, A.S., Georgeff, M.P.: Modeling rational agents within a BDI-architecture. In: Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91). (1991) 473–484

12. Cohen, P.R., Levesque, H.J.: Intention is choice with committment. Artificial Intelligence **42**(2-3) (1990) 213–261
13. Wooldridge, M.: Reasoning about Rational Agents. MIT Press (2000)
14. Meyer, J.J.C., van der Hoek, W., van Linder, B.: A logical approach to the dynamics of commitments. Artif. Intell. **113**(1-2) (1999) 1–40
15. Alechina, N., Logan, B.: A logic of situated resource-bounded agents. Journal of Logic, Language and Information **18**(1) (2009) 79–95
16. Mascardi, V., Ancona, D., Bordini, R.H., Ricci, A.: Cool-agentspeak: Enhancing agentspeak-dl agents with plan exchange and ontology services. In Boissier, O., Bradshaw, J., Cao, L., Fischer, K., Hacid, M.S., eds.: Proceedings of the 2011 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT 2011, Campus Scientifique de la Doua, Lyon, France, August 22-27, 2011, IEEE Computer Society (2011) 109–116
17. Bulling, N., Hindriks, K.V.: Taming the complexity of linear time bdi logics. In Sonenberg, L., Stone, P., Tumer, K., Yolum, P., eds.: 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011), Taipei, Taiwan, May 2-6, 2011, Volume 1-3, IFAAMAS (2011) 275–282
18. Alechina, N., Behrens, T., Dastani, M., Hindriks, K., Hubner, J., Logan, B., Nguyen, H., van Zee, M.: Multi-cycle query caching in agent programming. In: Proceedings of the Twenty-Seventh AAAI Conferernce on Artificial Intelligence (AAAI 2013), Bellevue, Washington, AAAI, AAAI Press (July 2013) (to appear).
19. Doyle, J.: A truth maintenance system. Artificial Intelligence **12**(3) (1979) 231–272
20. Forgy, C.: Rete: a fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence **19**(1) (1982) 17–37
21. Miranker, D.P.: TREAT: A better match algorithm for AI production systems. In: Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI'87), AAAI Press (1987) 42–47
22. van Riemsdijk, M.B., Dastani, M., Meyer, J.J.C.: Goals in conflict: semantic foundations of goals in agent programming. Autonomous Agents and Multi-Agent Systems **18**(3) (2009) 471–500
23. Thangarajah, J., Padgham, L.: Computationally effective reasoning about goal interactions. J. Autom. Reasoning **47**(1) (2011) 17–56
24. Thangarajah, J., Sardiña, S., Padgham, L.: Measuring plan coverage and overlap for agent reasoning. In van der Hoek, W., Padgham, L., Conitzer, V., Winikoff, M., eds.: International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012 (3 Volumes), IFAAMAS (2012) 1049–1056
25. Shapiro, S., Sardiña, S., Thangarajah, J., Cavedon, L., Padgham, L.: Revising conflicting intention sets in bdi agents. In van der Hoek, W., Padgham, L., Conitzer, V., Winikoff, M., eds.: International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012 (3 Volumes), IFAAMAS (2012) 1081–1088
26. Russell, S.J., Norvig, P.: Artificial Intelligence - A Modern Approach (3. internat. ed.). Pearson Education (2010)
27. Abrahamson, K.R.: Decidability and expressiveness of logics of processes. PhD thesis, Department of Computer Science, University of Washington (1980)
28. Mayer, A.J., Stockmeyer, L.J.: The complexity of PDL with interleaving. Theoretical Computer Science **161**(1&2) (1996) 109–122
29. Bordini, R.H., Bazzan, A.L.C., de Oliveira Jannone, R., Basso, D.M., Vicari, R.M., Lesser, V.R.: Agentspeak(xl): efficient intention selection in bdi agents via decision-

theoretic task scheduling. In: The First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002, July 15-19, 2002, Bologna, Italy, Proceedings, ACM (2002) 1294–1302
30. Thangarajah, J., Padgham, L.: An empirical evaluation of reasoning about resource conflicts. Proceedings of the Third International Conference on Autonomous Agents and Multiagent Systems **3** (2004) 1298–1299
31. Vikhorev, K., Alechina, N., Logan, B.: Agent programming with priorities and deadlines. In Turner, K., Yolum, P., Sonenberg, L., Stone, P., eds.: Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011), Taipei, Taiwan (May 2011) 397–404
32. Alechina, N., Dastani, M., Logan, B.: Programming norm-aware agents. In Conitzer, V., Winikoff, M., Padgham, L., van der Hoek, W., eds.: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012). Volume 2., Valencia, Spain, IFAAMAS (June 2012) 1057–1064