# Evaluating Coverage Based Intention Selection

Max Waters
RMIT University
Melbourne, Australia
max.waters@rmit.edu.au

Lin Padgham
RMIT University
Melbourne, Australia
lin.padgham@rmit.edu.au

Sebastian Sardina
RMIT University
Melbourne, Australia
sebastian.sardina@rmit.edu.au

## ABSTRACT

The Belief Desire Intention (BDI) agent paradigm provides a powerful basis for developing complex systems based on autonomous intelligent agents. These agents have, at any point in time, a set of intentions, the various tasks the agent is working on which represent the agent's multiple focus of attention. Despite its importance for intelligent behaviour, the problem of selecting *which intention to progress* at any point in time has received almost no attention and has been left to the programmer to resolve in an application-dependent manner. In this paper we implement and evaluate a previous proposal for domain-independent intention selection using the notion of plan "coverage," as well as a slight variation which we predicted to perform better. We compare these with the commonly used intention selection mechanisms of *First-In-First-Out* (FIFO) and *Round Robin* (RR). We show that the coverage-based technique performs better under *all* circumstances, but particularly with low coverage and volatile environments. Interestingly, we found that a simple one-step look-ahead applicability check is responsible for the largest part of the improvement. This is important in that this can readily be applied to FIFO and RR, giving an extremely simple and effective mechanism to be added to existing BDI frameworks.

## Categories and Subject Descriptors

I.2.5 [**Computing Methodologies**]: Artificial Intelligence—*Programming Languages and Software*

## General Terms

Algorithms, Experimentation, Performance

## Keywords

Intention selection, agent reasoning, BDI agents

## 1. INTRODUCTION

This work is concerned with the important problem of *intention selection* in intelligent agent systems. Intelligent autonomous agents are expected to be able to act appropriately in complex dynamic environments, to achieve their overall goals. The Belief Desire Intention (BDI) paradigm and associated programming languages and implemented toolkits are amongst the most successful approaches to realising smart autonomous agents. Under such a paradigm, the agent's behavior arises from its intentions, the current set of tasks that the agent has committed to realize—the agent's focus of attention. As agents will, in general, pursue multiple intentions, a great deal of the power of the paradigm relies on the mechanisms for selecting *which* intention to focus on at any particular step or point in time. (An equally important deliberation aspect is how to advance the selected intention, depending on various situational factors.) However, the mechanisms generally supported by typical BDI infrastructures for intention selection are very simplistic. Many agent platforms, such as Jack [6], offer a choice between *Round Robin* (RR), which does a fixed number of steps on each intention in turn, or a *First-In-First-Out* (FIFO) queue, which processes each intention in the order received, moving it to the back of the queue if for some reason it suspends. Some systems such as JAM [11] also allow priorities or utilities on goals and/or plans for ordering the intention queue.

Some authors (e.g., [2, 20, 10, 22]) have investigated the scheduling of intentions based on temporal information, such as either the time a goal must be achieved by, and/or an estimate of the time plans or goals take to execute. However, this information is generally not available in BDI programs written in popular BDI languages such as Jack, Jadex [13] or Jason [4]. Requiring programmers to add such information is cumbersome and often problematic. Thangarajah et. al [19] suggested a generic mechanism requiring no additional programmer-provided information. This was based on compile-time calculations of plans' *coverage* which, at the basic level, represents the percentage of world states which have some applicable plan for any subgoal within an intention. They defined how this measure could be extended to capture the inherited effects of lack of coverage of subgoals. They then specified how this could be used to (amongst other things) focus on potentially vulnerable intentions that were currently progressable. This mechanism is appealing in that it is application-independent and can be accommodated into the agent execution infrastructure to ensure that agents deliberate "intelligently" without the need for this to be explicitly programmed by the developer. Unfortunately though, the approach had not been implemented or evaluated and hence its actual benefits cannot be concretely stated.

In this work, we empirically evaluate the coverage-based intention selection approach (and a slightly improved variation), and compare it experimentally with both FIFO and single-step RR. Encouragingly, the proportion of successfully completed intentions increases by 60 (FIFO) and 62 (RR) percentage points, in volatile environments, where the plan library contains significant gaps in coverage. More importantly, in further analysis and experimentation, we identify a particular aspect of the approach which is responsible for the majority of the improvement, and can be incorporated into standard FIFO and RR with minimal effort.

**Appears in:** *Alessio Lomuscio, Paul Scerri, Ana Bazzan, and Michael Huhns (eds.), Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014), May 5-9, 2014, Paris, France.*

We first provide some background on the problem of agent intention selection, including an overview of the coverage-based approach described in [19]. We then describe the details of the experiments run (including the structure of the programs used), followed by the results. In Section 5 we analyse two separable aspects of the coverage-based selection approach, and conclude that, surprisingly, the major part of the gain comes from a simple progressability check as part of the intention selection. This is then applied to both FIFO and RR, giving gains of up to 48 percentage points.

# 2. THE INTENTION SELECTION TASK

BDI agent-oriented programming is a popular, well-studied, and practical paradigm for building intelligent agents situated in complex and dynamic environments with (soft) real-time reasoning and control requirements [1, 9]. Besides its philosophical roots in practical reasoning [5] and theoretical understandings [7, 15, 17], there is a plethora of BDI-style programming languages and systems available, such as Jack, Jason, Jadex, 2APL, and GOAL [3, 8].

A typical BDI-style agent system is depicted in Figure 1(a). An agent consists of a *belief base* $B$ (the agent's knowledge about the world), a set of recorded pending *events* or *goals*, a *plan library* (encoding the typical operational procedures of the domain), and an *intention base* (storing the plans/programs that the agent has already committed to and is executing).

Almost all BDI systems realize, in one way or another, the basic abstract rational interpreter described by Rao and Georgeff [16] and Bratman et. al's [5] IRMA rational architecture. In a nutshell, the BDI execution engine in Figure 1(a) responds to events—the inputs to the system—by *selecting* some of the pending events to handle (i.e., to respond to), *selecting* adequate operational programs from the plan library for handling such events, and *selecting* some current intentions to advance, that is, to actually execute. There are thus three important *choice points* in a rational agent framework. In this work, we are concerned with the latter one, namely, *how to select which intention to advance next at a given point*.

At every step along the execution of an agent system, the intention base represents the different focuses of attention the system has in order to respond to those events that the agent has committed to address [5]. *The overarching objective is for the system to be successful in carrying out all those intention programs to completion, thus resolving the corresponding events*. A problem, though, is that some such programs may end up failing because their execution context changes in unexpected ways. This could happen due to changes in the environment or due to negative interactions among the various executing intentions.

To tackle various contexts and unexpected changes, it is desirable for a BDI plan library to contain, for each event of concern, a selection of different strategies which covers as wide a range of potential situations as possible. For example, an aircraft controller may include landing procedures for different weather conditions. However, in most realistic scenarios, it is not feasible, or even possible, to provide concrete strategies for every possible situation. Like any knowledge, the agent's know-how will be intrinsically incomplete and have "gaps." If the agent is able to intelligently address its intentions when the situation for doing so matches its know-how, the chances of success will be improved.

The problem of *intention selection* is at the core of the BDI approach: *which intention should the agent select next for execution?* Together with the deliberation mechanisms for event and plan selection, it provides the actual "intelligence" of the whole framework. Unfortunately, however, the problem has been little studied and solutions are either extremely simplistic or are reliant on application-specific programming to provide the desired control.

A recent paper [19] suggested an interesting domain-independent approach to reasoning about intention selection, based on information which is already available in the BDI program, or can be computed at compile-time. It is this approach, based on calculations regarding gaps in agent "know-how" or *coverage* which we implement and then evaluate in Sections 3 and 4.

## 2.1 Coverage-based Intention Selection

The basic idea of the intention selection process proposed in [19], is to select, at every point, the intention that appears most "vulnerable," i.e., the one with the smallest executable context. The vulnerability of intentions is assessed using a refined version of plan coverage as introduced for Agent Oriented Software Engineering [21]. Coverage denotes the completeness of the know-how available to address a given event-goal, and is calculated in [19] using a model counter reasoning over the space of the context conditions. The lower the coverage for a goal, the less know-how there is available for different situations and the more vulnerable the event is to failure. The intention scheduler proposed in [19] basically prioritises intentions with low coverage, the intuition being that they should be executed immediately while the opportunity exists.

In order to explain how the coverage calculations of [19] work, we first describe the basic structure of BDI programs and the *goal-plan trees* induced by the plan library.

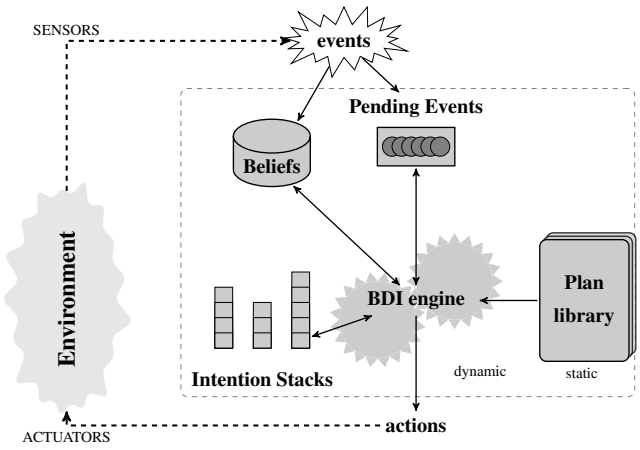
### *Goal-Plan trees*

Technically, a *plan* in the plan library is a rule of the form $G : \psi \leftarrow \delta$, meaning that program $\delta$ is a "reasonable strategy" to resolve event-goal $G$ *whenever the context condition* $\psi$ *is believed to be true*. Among other operations, program $\delta$ typically includes the execution of *primitive actions* (*act*) in the environment and the "posting" of new *subgoal events* ($!G$) that ought to be resolved by selecting (other) suitable plans. A plan may be selected for addressing an event if it is *relevant* and *applicable*, that is, if it is a plan designed for the event in question and its context condition is believed true. In contrast with traditional planning, execution happens at each step. The assumption is that the use of plans' context preconditions to make choices as late as possible, together with the built-in goal-failure mechanisms, ensures that a successful execution will eventually be obtained while the system is sufficiently responsive to changes in the environment.

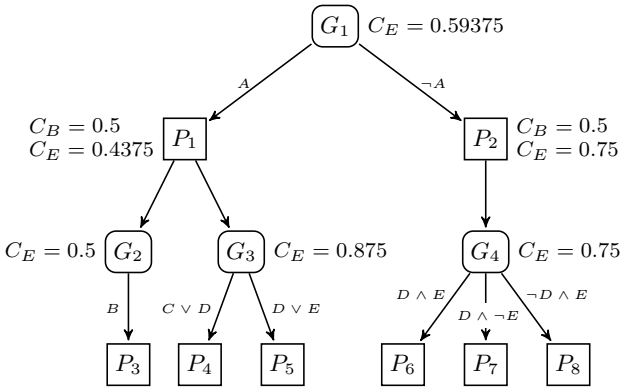
By grouping together plans responding to the same event type, the plan library induces *goal-plan tree* templates of the sort shown in Figure 1(b): a goal (or event) node (e.g., event-goal $G_1$) has children representing the alternative relevant plans for achieving it (e.g., plans $P_1$ and $P_2$), which, in turn, have children nodes representing the subgoals (including primitive actions) of the plan (e.g., subgoals $G_2$ and $G_1$ in plan $P_1$). These structures can be seen as AND/OR trees: for a plan to succeed all the subgoals and actions of the plan must be successful (AND); for a subgoal to succeed one of the plans to achieve it must succeed (OR). Leaf plans (e.g., $P_3$ to $P_8$) include no subgoals, but only primitive actions.

### *Coverage calculations*

In [19], the authors proposed a method to measure the coverage of plans and goals, including the idea of inherited or propagated coverage in the subtree below a particular goal. Firstly, the *basic coverage* $C_B$ of a plan is the fraction of the state space where the plan is applicable, that is, where it can be used. This can be calculated based on the plan's context condition. While [19] uses a model counter to calculate this, we use simplified representations where the coverage can be directly extracted. For example, plan $P_2$ has a 0.5 (or 50%) coverage: its context condition states that the




(a) A typical BDI-style architecture [17].




(b) Goal-plan hierarchy for goal $G_1$, with context conditions (edges' labels) and basic and extended coverage measures ($C_B$ and $C_E$).

**Figure 1: A BDI agent system architecture and a goal-plan tree hierarchy implicit in the plan library.**

plan is applicable in all worlds where proposition $A$ holds true.[1] Because both $D$ and $E$ ought to be true for plan $P_6$ to be applicable, its basic coverage amounts to 0.25 (or 25%). The coverage of a goal is essentially the sum of the coverage of its plans.[2] However the basic coverage of plans is not sufficient as it does not consider the fact that BDI plans will most often include subgoals, and any compromise in the coverage of a subgoal will affect the vulnerability of the higher level goal. Consequently an *extended coverage* $C_E$ is defined which takes into account the whole goal-plan hierarchy. The extended coverage of a plan is then calculated by multiplying its basic coverage by the product of the coverage of its subgoals (as all must succeed for the plan to succeed). The extended coverage of a goal is obtained by summing the extended coverage of all relevant plans (with the necessary adjustments for overlap). Figure 1(b) shows the basic coverage of plans, and the extended coverage of plans and goals.

### *Intention scheduling*

We now describe the coverage-based intention scheduler proposed in [19], which we refer to as $\mathcal{C}_0$, and a slight variation which we propose should be better. Essentially, $\mathcal{C}_0$ maintains focus as long as possible, reconsidering only when the current intention finishes, blocks (e.g., while awaiting a message response), or becomes unprogressable (i.e., posts a sub-goal with no applicable plans). When it does change focus it does so to the progressable intention with lowest coverage. Thus its first priority is progressability, its second is maintaining the current focus, and its third is execution of low coverage intentions. It is domain-independent and suitable for integration into any BDI-style system.

However, $\mathcal{C}_0$ does not appear to take maximum advantage of opportunities to execute vulnerable intentions. We propose a slight variation, which we will refer to as $\mathcal{C}_1$. This is a preemptive variation of $\mathcal{C}_0$ which emphasises the "opportunistic" prioritization of low-coverage intentions. Whenever a sub-goal is posted, each intention is re-assessed to calculate its coverage, and determine its progressability, that is, whether its current sub-goal has any applicable plans. The scheduler's first priority is towards progressability (i.e., the existence of an applicable plan), and its second priority is to execute intentions with low coverage, even if it means suspending the current intention. This preemption makes the scheduler more responsive to changes in the environment. If an environmental change means that an intention with low coverage which previously had no applicable plans can now be progressed, then the scheduler can take the opportunity to change focus and execute the lower coverage intention while the chance is there. Similarly, the scheduler can respond to any changes in the coverage of the currently executing intention. As an intention's plans are executed, its coverage will change. The preemption allows the scheduler to change focus to a lower coverage intention once, for example, a plan relying on a p-effect (see Section 3) has been executed.

We now explain our experimental comparison of $\mathcal{C}_0$ and $\mathcal{C}_1$ with both each other and with FIFO and RR.

## 3. EXPERIMENTAL SETUP

We have run a large number of experiments using automatically generated goal-plan trees with different coverage levels, executed in environments with varying dynamism. We describe the details of these experiments, in particular, *(a)* the nature and form of the goal-plan trees used and the source of coverage "gaps" on such trees; *(b)* the way a dynamic environment is modelled; and *(c)* the overall agent execution setup.

### *Goal-plan trees, p-effects, and coverage gaps*

The goal-plan trees generated automatically for our experiment resemble a binary decision tree, where each goal is handled by two plans, one requiring a particular variable in its context condition, and the other requiring its negation. Each plan is either a leaf node or posts exactly one subgoal. Note that the restriction to binary trees simplifies the automatic generation and control of such tree structures substantially, and is not a problem in itself, as single propositions may stand for complex formulae that are abstracted out. The simplification also means that we can calculate coverage directly rather than using a model counter as in [19].[3] The depth

[1]We assume here equal likelihood of $A$ and $\neg A$.

[2]We note that because context conditions of relevant plans (for a given goal event) may overlap, the degree to which the various plans overlap also needs to be calculated. Though not needed to understand this paper, we refer the reader to [19] for details.

[3]There are of course simplifications involved in having a single goal, and in having no overlap between plans, but we believe these

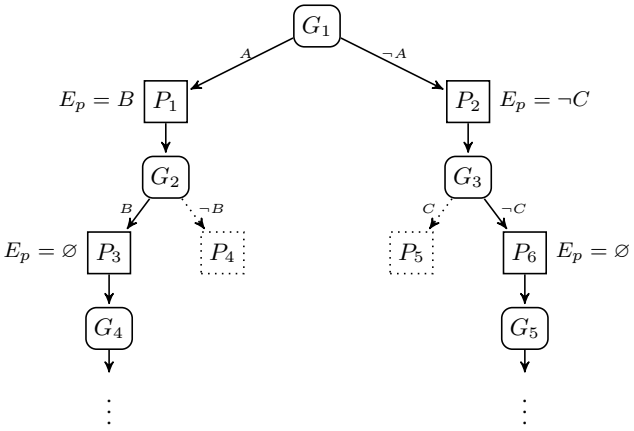



**Figure 2: A goal-plan structure for a goal $G_1$ with plans annotated with their p-effects $E_p$.**

of a goal-plan tree is the maximum number of goals from the root goal to a leaf plan (e.g., the tree in Figure 1(b) has a depth of two).

Probably the most important issue is the nature and form of the coverage "gaps" in the goal-plan tree structures used. Agent Oriented design methodologies, such as Prometheus [21], recommend that agent program developers carefully consider any incomplete goal coverage in their design. Nevertheless, it is common for developers to write programs so that the agent sets up the context conditions of plans that are meant to address a subsequent subgoal. These "conditions preparing for later steps" are known as *preparatory effects* or just *p-effects*. In the authors' experience, the reliance on p-effects is a typical cause of lack of complete coverage in agent programs.[4] The assumption is that once set up, those conditions will not be threatened, and therefore there is no need to provide plans (for the subgoal in question) for situations in which those conditions do not hold, hence creating a coverage "gap."

For example, in the goal-plan tree structure shown in Figure 2, there is no plan for subgoal $G_2$ in situations where proposition $B$ is false (i.e., no relevant plan for $G_2$ with context condition $\neg B$). However, an effect of plan $P_1$ is to make $B$ true, thus "preparing" the conditions for plan $P_3$ to apply for resolving $G_2$. The *setup distance* is the number of subgoals between the setting of the p-effect and its use in the context condition of a subsequent plan (one in our example).

Of course, in the absence of any environmental changes *and* any interleaving of the agent's intentions, the lack of coverage due to reliance on p-effects will have no adverse effect, as plans will be "chained" as the developer assumed. Indeed, under such (strong) assumptions, plan $P_3$ will be applicable and ready to execute, and the agent will experience no problems from the lack of coverage for goal $G_2$. However, agents are intended to operate in dynamic environments and should be able to interleave intentions.

In order to model coverage gaps based on p-effects, the basic binary structure initially generated is modified to create trees with less than full coverage by removing one of the plans handling a goal (e.g., removing plans $P_4$ and $P_5$ in Figure 2), and adding an appropriate p-effect into one of the goal's ancestors (e.g., adding $B$ and $\neg C$ as a p-effects of $P_1$ and $P_2$, respectively). Removing a plan in this way reduces the basic coverage of the corresponding goal (goals $G_2$ and $G_3$) from $C_B = 1.0$ to $C_B = 0.5$.

simplifications are justifiable and are discussed in Section 6.

[4]Work has been done to recognise p-effects and ensure that the agent does not itself undo them prematurely [18].

In order to be able to set up more specific and fine-grained coverage gaps, we use special propositions in context conditions that are sampled using a particular distribution. Conceptually this can be seen as equivalent to "merging" multiple plans/branches into a single representation. For example, consider goal $G_4$ and its three plans in Figure 1(b), with an extended coverage of $0.75$. A simplified version of this sub-tree can be obtained by merging plans $P_6$, $P_7$, and $P_8$ into a single plan $P_{678}$ whose context condition refers to a single "synthetic" proposition, say $X$, to be sampled true with probability $0.75$. In this way, we can conveniently and automatically generate goal-plan tree structures with arbitrary gap sizes.

There are of course other causes of incomplete coverage, such there being simply no way to achieve a goal from a particular world state. However, because reliance on p-effects is a common source of coverage gaps, we will focus on them here.

### *The environment and the agent's beliefs*

As any BDI agent, our agents track (the state of) the environment using a set of *belief propositions*, which are then used in the agent's plans as part of their context conditions for on-the-fly decision making (see Section 2). Because the external environment the agent is situated in is dynamic, we expect those propositions to sometimes change unexpectedly (that is, without the intervention of the agent). How frequent those unexpected changes are is determined by the so-called *dynamism rate* $d$ of the environment (where $d \in [0, 1]$, with $d = 1$ being very dynamic and $d = 0$ being fully static).

In our experiments, an *environment variable* is a tuple $\langle p, s, b\rangle$ comprised of a belief proposition $p$, a sampling probability $s \in [0, 1]$, and a boolean value $b \in \{\mathit{true}, \mathit{false}\}$ representing its current state. Whenever a subgoal is posted by the agent, random changes are applied to the environment based on the dynamism rate $d$. More concretely, any given environment variable $\langle p, s, b\rangle$ is re-sampled with probability $d$, and if it is re-sampled, then the current state $b$ is set to *true* with probability $s$ (or *false* with probability $1 - s$).

### *Testbed setup*

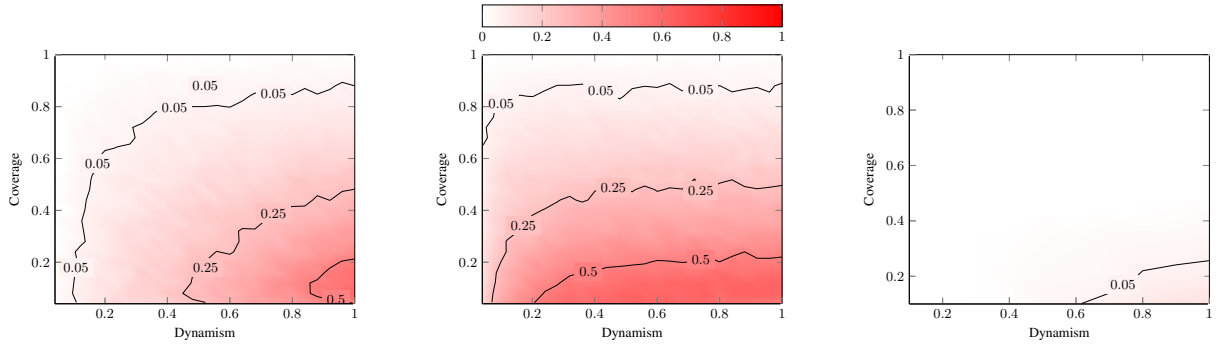
We ran agents with ten intentions, each being a goal-plan tree with a depth of five, a setup distance of one, and two places with coverage gaps (i.e., goals whose plans do not cover the whole state space).

Each test run is set up with *(i)* a randomly selected dynamism level $d$; *(ii)* a sample of the environment's initial state, generated using the sampling probabilities of each proposition; and *(iii)* a randomly selected average coverage for the agent's goal-plan trees. As explained above, the sampling probability distributions of the relevant environment variables are adjusted to ensure that the agent's goal-plan trees have the required coverage.

With the above settings as parameters to test runs, each of the four intention selection algorithms, namely, $\mathcal{C}_0$ $\mathcal{C}_1$, RR and FIFO, were tested and their success rates—the proportion of intentions that are successfully carried out to completion—recorded. A total of $100,000$ tests were generated and tested with each algorithm. To compare two algorithms, we took the difference between their success rates, as measured in *percentage points* (pp). For example, if scheme $A$ has a success rate of $0.3$ (i.e., $30\%$ of intentions complete) and scheme $B$ has a success rate of $0.4$ (i.e., $40\%$), then we say that $B$ improves on $A$ by $0.1$, or $10$pp.

## 4. COVERAGE-BASED SELECTION

Figure 3 depicts the first experimental results obtained when comparing intention scheduling approach $\mathcal{C}_1$ with FIFO, RR, and $\mathcal{C}_0$. The $x$ and $y$ axes range over the environmental dynamism rate $d$ (as $d$ approaches 1 the environment becomes more volatile) and the average coverage of the goal-plan trees used (as it approaches 1,




(a) $\mathcal{C}_1$ − FIFO (b) $\mathcal{C}_1$ − RR (c) $\mathcal{C}_1$ − $\mathcal{C}_0$

**Figure 3: A comparison of the success rates of $\mathcal{C}_0$, $\mathcal{C}_1$, FIFO and RR. The colour indicates the difference in success rate achieved by the given algorithms, with 0 (white) indicating no difference and 1 (red) indicating an improvement of 100pp. The contour lines indicate the point at which the difference in success rate crosses a specified threshold.**

coverage is higher and hence programs more robust). The colour intensity indicates the difference in success rate achieved by $\mathcal{C}_1$ and each of the other three algorithms.

The overall conclusion is that $\mathcal{C}_1$ constitutes a significant improvement over both FIFO and RR, and there are no circumstances in which FIFO or RR have a statistically significant advantage over $\mathcal{C}_1$. As expected, such improvement gets more pronounced as the environment gets more dynamic and the coverage of goals decreases.

The first section of Figure 4(e) shows the combined test results, grouped by scheduling algorithm. The table displays, for each sample group, the mean ($\mu$) and standard deviation ($\sigma$) of the success rate $r$ as averaged over all levels of coverage and dynamism. Even though each algorithm was tested $10,000$ times, over a wide range of coverage and dynamism levels, the values of $\sigma_r$ for $\mathcal{C}_0$ and $\mathcal{C}_1$ are relatively low, suggesting that the samples are clustered tightly around $\mu_r$. Indeed, the groups were compared using a two-tailed, paired t-test, revealing that $\mathcal{C}_1$ improves upon $\mathcal{C}_0$ by 1pp, FIFO by 13pp and RR by 23.5pp, all with *p-value* $< 0.001$.

Figure 3(a) shows that there is no circumstance in which FIFO out-performs $\mathcal{C}_1$. In test cases which used either very robust programs or an almost static environment (i.e., as coverage approaches 1 or dynamism approaches 0), there is little to no difference between the success rate of the two algorithms ($<$ 1pp, or too small to be statistically significant). However, as the programs become less robust and the environment more volatile, the difference becomes more pronounced. Indeed, as coverage dips below 0.6 and dynamism rate exceeds 0.4, $\mathcal{C}_1$ improves upon FIFO by 15pp. In more extreme circumstances, differences of up to 60pp are observed.

Similarly, Figure 3(b) shows that $\mathcal{C}_1$ always improves on RR, in particular, under high environmental dynamism and low goal coverage areas, where improvements of up to 62pp are observed. Interestingly, and in a marked difference with FIFO, $\mathcal{C}_1$ out-performs RR *even in relatively static environments*. The fact is that, even in slightly dynamic environments, p-effects are more vulnerable when running RR as the intention scheduler. This is because RR will make the agent change focus (i.e., change intention) at every execution step, meaning that between a p-effect being enacted and the dependent plan being executed, every other intention will be executed at least once. As a result, there is more time for the environment to undo the changes made by the p-effect.

Finally, Figure 3(c) demonstrates that $\mathcal{C}_1$'s preemption scheme (see Section 2) gives it an advantage of up to 14pp over $\mathcal{C}_0$ when running poor-quality plan libraries in dynamic environments, though in most of the space there is no real difference between them.

# 5. APPLICABILITY CHECKING

There are some significant differences between the selection methods used by $\mathcal{C}_1$ and FIFO. FIFO stores all intentions in a queue, and simply steps through them one at a time, with no prioritisation beyond arrival time. $\mathcal{C}_1$ selects, at each step, the progressable intention with the lowest coverage. Only if all intentions are unprogressable will an intention with no applicable plans be selected. This will cause the current subgoal to fail, and either an alternative approach (which may have become applicable due to changes in the world) will be tried due to failure recovery, or the intention itself will fail as a whole. $\mathcal{C}_1$ is therefore doing more than just prioritisation—it is also performing a very simple form of "look-ahead." Indeed, $\mathcal{C}_1$ looks ahead by just one step in an intention, and attempts to foresee whether the next subgoal would fail (due to no applicable plan) if the intention were selected.

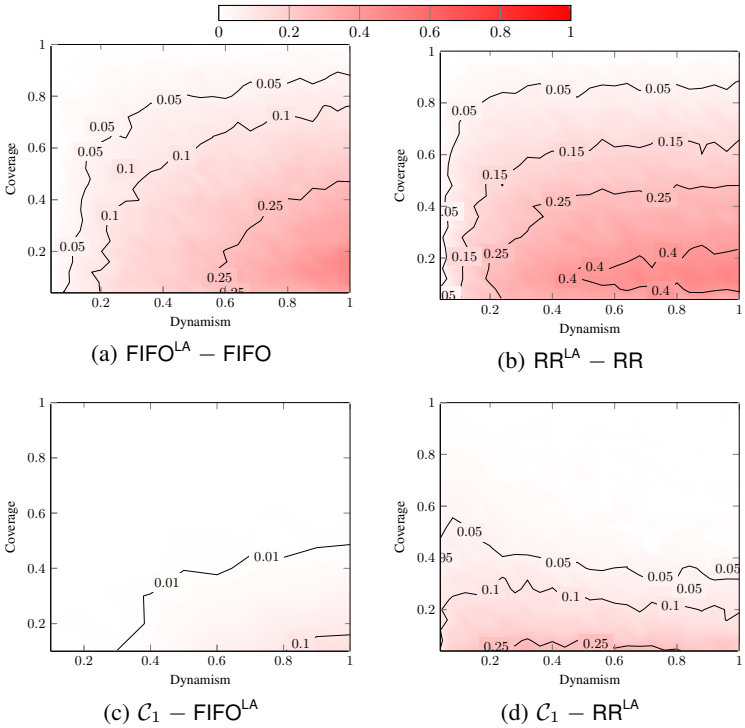
This raises two questions. Firstly, how much of $\mathcal{C}_1$'s advantage over FIFO and RR can be attributed to its look-ahead, rather than its prioritisation? Secondly, can FIFO or RR be improved by the inclusion of this simple look-ahead technique?

To answer these questions we define two *look-ahead variants* FIFO$^{\text{LA}}$ and RR$^{\text{LA}}$. Scheme FIFO$^{\text{LA}}$ selects the first progressable intention in its queue, and maintains focus until the intention becomes unprogressable. When all intentions are unprogressable, FIFO$^{\text{LA}}$ behaves as FIFO by selecting (and applying failure recovery to) the intention at the head of the queue. Like RR, the RR$^{\text{LA}}$ approach selects intentions cyclically, but skips over any unprogressable intentions. If all intentions are found to be unprogressable, it selects an intention (for failure recovery) as per normal RR behaviour.

These two intention selection variants were tested under the same experimental setup described in Section 3. Interestingly, and to our surprise, the experiments showed that *the use of look-ahead significantly improves the effectiveness of both FIFO and RR schemes*. This is important, as incorporating look-ahead into standard BDI scheduling algorithms is (almost) trivial.

Figure 4(e) shows the mean success rates of FIFO, RR, FIFO$^{\text{LA}}$ and RR$^{\text{LA}}$, averaged over all tests. Using the same t-test as in Section 4 reveals that look-ahead improves FIFO's performance by 12.6pp. Similarly, look-ahead in RR$^{\text{LA}}$ is responsible for an improvement of 18pp over RR. (Both with *p-value* $< 0.001$.)

Figures 4(a)–4(d) depict the effect of look-ahead in more detail. It is clear from Figure 4(a) that the use of look-ahead is never detrimental to the operation of FIFO: even on robust programs (coverage $> 0.8$) or under fairly static environments ($d < 0.2$), it produces a 5pp increase in its success rate. As coverage falls below 0.6,




(a) FIFO$^{\text{LA}}$ − FIFO

(b) RR$^{\text{LA}}$ − RR

(c) $\mathcal{C}_1$ − FIFO$^{\text{LA}}$

(d) $\mathcal{C}_1$ − RR$^{\text{LA}}$

| SCHEDULER | $\mu_r$ | $\sigma_r$ |
|---|---|---|
| Setup distance $s = 1$ | | |
| $\mathcal{C}_0$ | 0.953 | 0.098 |
| $\mathcal{C}_1$ | 0.963 | 0.079 |
| FIFO | 0.825 | 0.204 |
| RR | 0.728 | 0.256 |
| FIFO$^{\text{LA}}$ | 0.951 | 0.101 |
| RR$^{\text{LA}}$ | 0.909 | 0.144 |
| Setup distance $s = 2$ | | |
| $\mathcal{C}_0$ | 0.936 | 0.121 |
| $\mathcal{C}_1$ | 0.952 | 0.095 |
| FIFO | 0.781 | 0.238 |
| RR | 0.729 | 0.272 |
| FIFO$^{\text{LA}}$ | 0.935 | 0.125 |
| RR$^{\text{LA}}$ | 0.882 | 0.168 |
| Setup distance $s = 3$ | | |
| $\mathcal{C}_0$ | 0.931 | 0.131 |
| $\mathcal{C}_1$ | 0.948 | 0.101 |
| FIFO | 0.775 | 0.246 |
| RR | 0.746 | 0.274 |
| FIFO$^{\text{LA}}$ | 0.928 | 0.137 |
| RR$^{\text{LA}}$ | 0.870 | 0.176 |

(e) Mean ($\mu$) and standard deviation ($\sigma$) of the success rate $r$, averaged over all levels of coverage and dynamism.

**Figure 4: The effect of look-ahead on FIFO and RR scheduling (4(a) and 4(b)); a comparison of the success rates of $\mathcal{C}_1$ with FIFO$^{\text{LA}}$ and RR$^{\text{LA}}$ (4(c) and 4(d)); and success rates for all schedulers on three setup distances (4(e)).**

improvements of 15pp are obtained. In extreme cases, where the dynamism rate approaches 1 and coverage is 0.1, an improvement of 48pp is seen. Similarly, Figure 4(b) shows that look-ahead never hinders the performance of RR, but rather produces improvements of over 40pp, even in relatively static environments.

By comparing $\mathcal{C}_1$'s results with look-ahead based algorithms, it is possible to isolate the benefit provided by $\mathcal{C}_1$'s prioritization. A t-test reveals that, on average, $\mathcal{C}_1$ is more successful than FIFO$^{\text{LA}}$ and RR$^{\text{LA}}$, but by merely 1.2pp and 5.3pp, respectively (both with *p-value* $< 0.001$).

Nonetheless, Figures 4(c) and 4(d) suggests that there may be situations in which it is still preferable to use $\mathcal{C}_1$ over FIFO$^{\text{LA}}$ or RR$^{\text{LA}}$. While prioritizing by coverage has little benefit for robust programs (coverage $> 0.5$), it can in fact provide a benefit of up to 14pp over FIFO$^{\text{LA}}$ and 35pp over RR$^{\text{LA}}$ when running fragile programs in adverse conditions. Further to this, there are no circumstances in which prioritizing low coverage intentions is detrimental to the operation of the program.

### *Many interacting intentions*

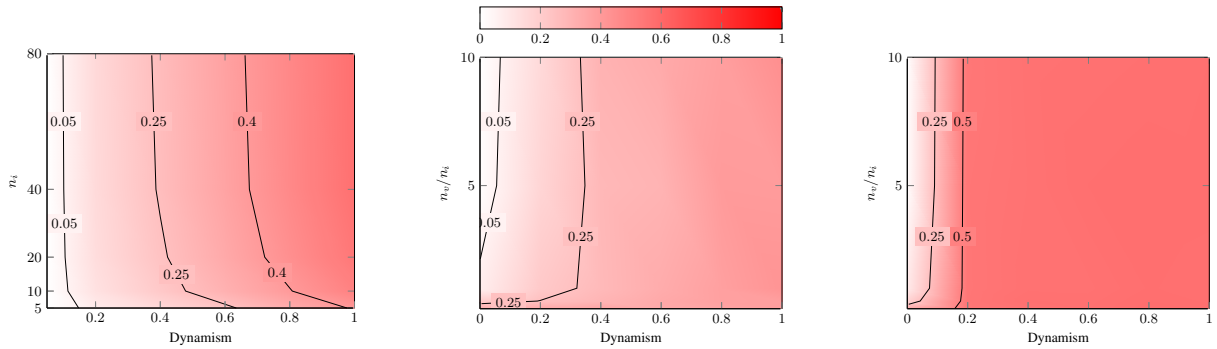
As stated in Section 3, agents in the above experiments had ten concurrent intentions, as it is unlikely an agent would be managing more than this number of parallel tasks of different types. However, many more intentions may be required in applications where an agent may have multiple *instances* of a single type of task, such as in a warehouse or factory. Consequently, we have explored the value of FIFO$^{\text{LA}}$ and RR$^{\text{LA}}$ in such situations.

The goal-plan trees for these experiments were produced in the same way as described before (see Section 3), but now using structurally identical trees (to represent different instances of the same intention). Also, we allow them to refer to the same domain variables to capture that such intentions will typically have the potential to conflict, as they could potentially change the same aspects of the environment and require access to the same resources.

Before each test run, the variables referred to by the context conditions of each plan across every goal-plan tree are selected (with a 50% chance of negation) from a pool of a fixed size $n_v$. Clearly, the fewer variables ($n_v$), or the greater the number of goal-plan trees ($n_i$), the more likely it is that a variable will occur in more than one goal-plan tree. Therefore the amount of interaction between the intentions is inversely proportional to $n_v/n_i$—the lower this value, the more interaction. By varying the values of $n_i$ and $n_v$ before each test run, the amount of interference can be controlled.

Because the goal-plan trees are structurally identical, and all variables have the same distribution, all of the goal-plan trees have the same initial extended coverage. Therefore, we have examined only the gains from the look-ahead mechanism, performing $40,000$ test runs on FIFO, FIFO$^{\text{LA}}$, RR, and RR$^{\text{LA}}$, with varying levels of dynamism, variable pool sizes $n_v$, and number of goal-plan trees $n_i$.

As would be expected, FIFO is affected not by the number of intentions or the amount of interaction between them, but only by the dynamism of the environment. However, FIFO$^{\text{LA}}$ is affected by both the dynamism rate and the number of concurrent intentions. Figure 5(a) compares their performances under different values for these two variables. While both FIFO and FIFO$^{\text{LA}}$ achieve a success of 100% in static environments, the use of look-ahead yields significant advantages in more dynamic situations. For example, when $n_i = 5$ and $d = 1.0$, FIFO$^{\text{LA}}$ gives a success rate improvement of 42pp over FIFO; and when $n_i = 80$, FIFO$^{\text{LA}}$'s success reaches 99%, an improvement of 57pp over plain FIFO.




(a) FIFO$^{\text{LA}}$ − FIFO (b) RR$^{\text{LA}}$ − RR, $n_i = 5$ (c) RR$^{\text{LA}}$ − RR, $n_i = 80$

**Figure 5: The effect of look-ahead on FIFO and RR scheduling, with multiple interacting intentions.**

The increase in FIFO$^{\text{LA}}$'s success rate with the increase of the number of intentions is a direct consequence of the use of look-ahead. A look-ahead based scheduler prioritizes progressable intentions, i.e., failure recovery will only kick in if *all* intentions are stuck. The more intentions the scheduler has to choose from, the more likely it is that at least one will be progressable.

The RR and RR$^{\text{LA}}$ schedulers behave in more complex ways—their success rates are affected by the amount of interaction, the dynamism rate, and the number of intentions. Because RR constantly switches from task to task, it is more susceptible to failure due to tasks interfering with each other. Hence, its success rate drops as the ratio $n_v/n_i$ decreases. Figure 5(b) shows that the integration of look-ahead improves on this. With low $n_v/n_i$ values, and a static environment, RR$^{\text{LA}}$ improves on RR by up to 32pp.

The negative results achieved by RR are exacerbated by any increase in the dynamism of the environment. Interestingly, RR$^{\text{LA}}$ thrives in such conditions. In a chaotic environment (dynamism $d = 1$) with closely related intentions ($n_v/n_i = 0.5$), as the number of concurrent intentions increases from 5 to 80, the success of RR$^{\text{LA}}$ increases from 76% to 99%. As shown in Figures 5(b) and 5(c), these results constitute, respectively, increases of 43pp and 57pp over RR. The explanation for this is the interaction between progressability and dynamism (arising both from the environment and interacting intentions). Once progressability checking is included to prevent unnecessary failure, the environmental dynamism also ensures frequent opportunities for success. As long as there are a sufficient number of intentions, something can almost always be successfully progressed, and an unprogressable intention will soon become progressable again.

From these results we conclude that integrating look-ahead into standard BDI intention selection strategies provides increased benefits as the number of concurrent (same type) intentions increases.

### *Setup distance effects*

As explained in Section 3, the setup distance $s$ is the number of subgoals posted between a p-effect and its dependent plan. Along with coverage, it is a measure of an intention's robustness—the greater the setup distance, the less robust the intention. While the experiments reported so far have $s = 1$, we have also experimented with other values.

To do so, we used the same experimental setup from Sections 4 and 5, with varying values for the dynamism rate, coverage as well as setup distance. We performed $50,000$ test runs for each algorithm $\mathcal{C}_0$, $\mathcal{C}_1$, FIFO, FIFO$^{\text{LA}}$, RR and RR$^{\text{LA}}$, and the results show that both coverage and look-ahead based selection methods are more able to handle higher setup distances.

The results are shown in Figure 4(e). Unexpectedly, RR's success rate actually *increases* with the setup distance. RR's lack of focus means that there is an increased opportunity for p-effects to be undone by random environmental changes. As the delay between a p-effect and its dependent plan increases (e.g., by increasing the setup distance), the chance of the p-effect holding decreases, and the likelihood of the dependent plan still being usable approaches the coverage measure of the plan's context condition. Increasing the delay beyond this convergence point has no further effect. RR is therefore affected only slightly by increasing $s$. However, a side-effect of increasing the setup distance $s$ is that the opportunity for failure recovery increases, due to dependent plans occurring deeper within the goal-plan tree. In the case of RR, this is enough to counteract the slight effect of increasing the setup distance.

Other than this one anomaly, all algorithms *perform worse as $s$ increases*, with those incorporating look-head ($\mathcal{C}_1$, FIFO$^{\text{LA}}$ and RR$^{\text{LA}}$) being affected slightly less. For example, as $s$ increases from 1 to 3, FIFO$^{\text{LA}}$'s success rate decreases by 2.3pp, while FIFO's success rate drops by 5pp. The use of coverage as a priority further enables a selection method to handle longer setup distances, e.g., $\mathcal{C}_1$'s success rate drops by just 1.5pp over the same distance range.

As before, more pronounced differences are seen under low coverage and/or high dynamism situations. In such circumstances and with $s = 1$, $\mathcal{C}_1$ has an advantage over FIFO$^{\text{LA}}$ of 14pp, increasing to 17pp with $s = 2$, and 20pp with $s = 3$. Similarly, $\mathcal{C}_1$'s advantage over RR$^{\text{LA}}$ increases from 35pp to 40pp and then to 45pp as $s$ increases. The difference between $\mathcal{C}_1$ and $\mathcal{C}_0$ increases from 12pp to 14pp and then 16pp.

## 6. DISCUSSION AND FUTURE WORK

In this paper, we have empirically analysed various intention selection mechanisms for BDI agent architectures, a key aspect of the BDI agent-oriented programming paradigm. Our experimentation clearly demonstrates that *the use of progressability checking when doing intention selection gives a substantial improvement* in the number of successfully completed intentions, and that the use of prioritisation based on coverage further improves this, especially in volatile environments where coverage is low. The substantial value of progressability checking is an important result, in that it is straightforward to incorporate this into any BDI language/system in the AgentSpeak [14] tradition. Indeed, the operations required to implement it (i.e., calculating the applicable set and testing context conditions) are standard elements of this family of agent languages.

One aspect of this approach is that having made a decision as to *how to* accomplish a particular (sub)goal, the agent is strongly committed to continuing with that approach, and is less responsive

to the changing situation. In practical applications this may have both advantages and disadvantages. Being fast to react, fail the subgoal, and look for alternative ways to accomplish the higher level intention (standard behaviour) can result in unnecessary wasted effort or wasted resources, so postponing progression (as we do via progressability checking) can be advantageous. However, if the intention is urgent, then waiting, rather than actively looking for an alternative solution, would be undesirable. If our agent was attempting to shut down a malfunctioning nuclear reactor, we would not want it to wait for the situation to change for the better, rather than fail the current approach and attempt some other plan. The natural response to this is simply to provide one or more mechanisms to allow flexibility regarding exactly when progressability checking is used. A combination with a simple priority scheme could ensure that urgent intentions are never paused. Alternatively, the agent could remain committed to the current approach, only until some alternative plan at a higher level in the goal-plan tree was known to be applicable, as described in [17]. The experimental work of Kinny and Georgeff [12] on the impact of agent commitment levels under different rates of change in the environment is orthogonal to ours, and it would be very interesting to extend our experiments to account for cautious and bold agents.

The goal-plan trees that we have used for this experimentation are clearly simplified as compared with real BDI programs. In particular one would virtually never have plans with a single subgoal. Also, the lack of overlap means that there is less opportunity for failure recovery than might be present in actual programs (depending on the design). However, these simplifications do not impact the ability to explore the effect of the intention selection mechanisms, and they do allow the use of large numbers of automatically generated intention structures. In future work we plan to explore the behaviour when this approach is used with a complex existing JACK program, to document and qualitatively explore actual differences in overall execution behaviour. It is however key to first understand the behaviour well on simplified structures.

The addition of generic reasoning mechanisms to improve the "intelligence" of intention selection is a crucial area of research for intelligent agent technology. It is also important from a practical point of view that such approaches do not require substantial additional information to be provided by the developer. Both the pure look-ahead approach as well as the coverage-based intention selection scheme that we have explored in this work meet this criteria. We expect that following the necessary qualitative exploration with some complex existing programs, at least the $\mathsf{FIFO}^{\mathsf{LA}}$ and $\mathsf{RR}^{\mathsf{LA}}$ algorithms might be integrated into existing BDI agent platforms.

## Acknowledgements

We acknowledge the support of the Australian Research Council under Discovery Project DP1094627. We also thank Agent Oriented Software for providing us with a Jack license.

## 7. REFERENCES

[1] S. S. Benfield, J. Hendrickson, and D. Galanti. Making a strong business case for multiagent technology. In *Proc. of Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 10–15, 2006.

[2] R. H. Bordini, A. L. C. Bazzan, R. de Oliveira Jannone, D. M. Basso, R. M. Vicari, and V. R. Lesser. AgentSpeak(XL): Efficient intention selection in BDI agents via decision-theoretic task scheduling. In *Proc. of Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1294–1302, 2002.

[3] R. H. Bordini, L. Braubach, M. Dastani, A. Fallah-Seghrouchni, J. J. Gómez Sanz, J. Leite, G. O'Hare, A. Pokahr, and A. Ricci. A survey of programming languages and platforms for multi-agent systems. *Informatica (Slovenia)*, 30(1):33–44, 2006.

[4] R. H. Bordini, J. F. Hübner, and M. Wooldridge. *Programming Multi-agent Systems in AgentSpeak Using Jason*. Wiley, 2007. Wiley Series in Agent Technology.

[5] M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(3):349–355, 1988.

[6] P. Busetta, R. Rönnquist, A. Hodgson, and A. Lucas. JACK intelligent agents: Components for intelligent agents in Java. *AgentLink Newsletter*, 2:2–5, Jan. 1999.

[7] P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.

[8] F. S. de Boer, K. V. Hindriks, W. van der Hoek, and J.-J. Meyer. A verification framework for agent programming with declarative goals. *Journal of Applied Logic*, 5(2):277–302, 2007.

[9] M. P. Georgeff and F. F. Ingrand. Decision making in an embedded reasoning system. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 972–978, 1989.

[10] B. Horling, V. Lesser, R. Vincent, and T. Wagner. The Soft Real-Time Agent Control Architecture. *Autonomous Agents and Multi-Agent Systems*, 12(1):35–92, 2006.

[11] M. J. Huber. JAM: A BDI-theoretic mobile agent architecture. In *Proc. of the Annual Conference on Autonomous Agents (AGENTS)*, pages 236–243, 1999.

[12] D. Kinny and M. P. Georgeff. Commitment and effectiveness of situated agents. In *Proc. of the Int. Joint Conference on Artificial Intelligence (IJCAI)*, pages 82–88 1991.

[13] A. Pokahr, L. Braubach, and W. Lamersdorf. JADEX: A BDI reasoning engine. In R. H. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 149–174. Springer, 2005.

[14] A. S. Rao. Agentspeak(L): BDI agents speak out in a logical computable language. In *Proc. of the European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, volume 1038 of *LNCS*, pages 42–55. Springer, 1996.

[15] A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In J. F. Allen, R. Fikes, and E. Sandewall, editors, *Proc. of Principles of Knowledge Representation and Reasoning (KR)*, pages 473–484, 1991.

[16] A. S. Rao and M. P. Georgeff. An abstract architecture for rational agents. In *Proc. of Principles of Knowledge Representation and Reasoning (KR)*, pages 438–449, 1992.

[17] S. Sardina and L. Padgham. A BDI agent programming language with failure recovery, declarative goals, and planning. *Autonomous Agents and Multi-Agent Systems*, 23(1):18–70, 2011.

[18] J. Thangarajah, L. Padgham, and M. Winikoff. Detecting & avoiding interference between goals in intelligent agents. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 721–726, 2003.

[19] J. Thangarajah, S. Sardina, and L. Padgham. Measuring plan coverage and overlap for agent reasoning. In *Proc. of Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1049–1056, 2012.

[20] K. Vikhorev, N. Alechina, and B. Logan. Agent programming with priorities and deadlines. In *Proc. of Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 397–404, 2011.

[21] M. Winikoff and L. Padgham. *Developing Intelligent Agent Systems: A Practical Guide*. John Wiley and Sons, 2004.

[22] H. Zhang, S. Y. Huang, and Y. Chang. An agent's activities are controlled by his priorities. In *Proc. of the KES International Conference on Agent and Multi-agent Systems: Technologies and applications*, pages 723–732, 2008.