Softgoal-based Plan Selection in Model-driven BDI Agents

Ingrid Nunes
Instituto de Informática
Universidade Federal do Rio Grande do Sul
Porto Alegre, Brazil
ingridnunes@inf.ufrgs.br

Michael Luck
Department of Informatics
King's College London
London, UK
michael.luck@kcl.ac.uk

ABSTRACT

The belief-desire-intention (BDI) model is one of the most widely used for developing agents. One of its benefits is the flexibility of choosing among different plans to achieve a goal and, to leverage this benefit, a particular algorithm that makes this choice must be selected. Although many techniques have been proposed addressing the plan selection process — as well as other aspects of BDI agents — they require many customisations and adaptations to be used in particular applications, thus requiring expert knowledge to be adopted, which is a real barrier to the large-scale adoption of this kind of agent technology. We thus in this paper propose a model-driven approach that allows modelling agents based on an extended BDI model (inspired by the Tropos meta-model) and automatically generating source code that implements agents with the ability of selecting plans. As this plan selection process typically involves analysing plan side effects, e.g. cost of execution, in the context of current agent preferences, our approach uses a preference-based plan selection process over what we refer to as softgoals. This process relies on the multi-attribute utility theory, taking into account the uncertainty of plan outcomes. We evaluate our approach experimentally.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Intelligent agents; D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*

General Terms

Design

Keywords

BDI Agent, Model-driven Development, Plan Selection

1. INTRODUCTION

The belief-desire-intention (BDI) model [19] is one of the most widely used for developing agents, and is supported by very many proposed methods and techniques that rely on it [3, 7, 14], and platforms that implement it [2, 10, 16, 18]. A key benefit of the BDI model is to provide a reasoning cycle that endows agents with flexible behaviour due to explicit separation between system goals and plans. However, this benefit is only obtained if a plan selection algorithm is chosen to suit a specific application, selecting appropriate plans according to particular situations. BDI agents typically use a plan library constructed at design time, so that at runtime agents must choose a plan to be executed from a set of applicable plans to achieve a goal, taking into account plan characteristics and agent beliefs, including preferences.

Many plan selection techniques have been proposed in the literature (e.g., [4, 21]) — as well as techniques addressing other customisable parts of BDI agents, such as belief revision and goal generation — but their direct use by mainstream software developers is still problematic, since expert knowledge is still required to develop BDI agents; this is a real barrier to the large-scale adoption of this kind of agent technology. It is important therefore not only to provide a conceptual basis for the development of BDI agents, but also to show that it is realisable (and how) and can be effectively used in real world applications. Without providing practical implementation, the use of proposed techniques can only be limited to a small set of specialists, preventing its use by the broader development community that could benefit from it. Yet this is crucial if we are to address the increasing complexity of software applications that must now be provided with flexible and intelligent behaviour.

A promising approach to raise the abstraction level of models when developing applications, thus making the use of sophisticated techniques easier through the abstraction of low-level details, is Model-driven Development (MDD) [20], which has previously been investigated to support the development of multi-agent systems (MASs) [5]. Such model-driven approaches use models as *first class citizens* in the development process, rather than code, and focus on building high-level models that can be used to eventually generate source code, with model compositions, and model-to-model and model-to-text transformations. However, although there has been much progress in the context of model-driven MASs, this is still premature as it currently permits the modelling of only *some* multi-agent concepts and only generates skeletons of source code. Therefore, much (and perhaps most of the) effort is still required to implement a MAS even after generating part of its code.

Our vision is to build model-driven MASs focusing on developing application-specific models and encoding MAS techniques into code generators, promoting faster application development with less expert knowledge about MAS techniques in order to effectively use them. In this paper, therefore, we propose a model-driven approach that takes a step towards this vision focusing on the plan selection process. Our approach consists of a meta-model that allows agents to be modelled, based on an extended BDI model — inspired by the Tropos [3] meta-model — and automatically generates source code that implements agents able to effectively select plans. As this plan selection process typically involves analysing

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.*

plan side effects, e.g. time to achieve a goal or cost of execution, in the context of current agent preferences, our approach uses a preference-based plan selection process over what we refer to as *softgoals*. This process relies on multi-attribute utility theory (MAUT) [13], taking into account the uncertainty of plan outcomes. To evaluate our work, we conducted an empirical experiment, consisting of a simulation of an agent built using our approach.

In summary, the contributions of this paper are: (i) a meta-model that extends the BDI model to allow representation of the concepts needed for agents able to select plans based on softgoals and preferences; (ii) a simple but effective algorithm to select plans using the concepts of our meta-model; and (iii) a technique to transform instances of our meta-model into source code.

In the next section (Section 2), we introduce an example that illustrates the scenario we are addressing. In Section 3, we present an extension of the BDI model with a set of concepts, which is used in the plan selection process. We then show in Section 5 how an agent based on our extended BDI model has its code generated via design models. In Section 6, we empirically evaluate our approach. Finally, Section 7 presents related work, followed by Section 8, which concludes.

2. ILLUSTRATING EXAMPLE

In this section, we present a motivating example that is used subsequently to explicate our approach. Consider the common problem of researchers in obtaining funding to attend conferences. *Bob* is a researcher at a university, who has three different options (or **plans**) to apply for and receive funding whenever he has a paper accepted at a conference. The options are: (i) *Plan A*: to apply for a continuous call for funding requests provided by the national funding agencies; (ii) *Plan B*: to request financial support from his department; and (iii) *Plan C*: to use the resources of his own research projects. An **agent** $BobA$ is able to automatically execute these plans.

Although all these plans are a means of achieving the **goal** of *getting funding for attending a conference*, each plan has different characteristics. First, each plan is associated with a particular response time so that, for example, if Bob chooses Plan A, he will be unsure whether and when he will receive approval of his request, whereas if he chooses Plan C (assuming that he has his own projects), he knows right away that he will be able to fund his trip. Second, each plan requires a different amount of work; e.g. requesting support to the department implies completing many forms as part of the request process. Third, each plan has a different impact on Bob's goal of minimising expenditure. For instance, if Bob uses his own resources, he will be unable to use them in the future for emergency situations, yet this is the plan with the highest probability of success. Therefore, all plans achieve a common goal, but also impact on other **softgoals**, namely *minimise time*, *minimise amount of work*, and *saving money*, which typically conflict, since increasing the satisfaction of one softgoal may decrease the satisfaction of another. Moreover, whenever Bob wants to achieve the goal *getting funding for attending a conference*, he has different **preferences**. For example, if he needs to confirm his attendance at the conference quickly, the softgoal *minimise time* is more important than others.

This scenario exposes the following problem: *how can agent $BobA$ select the best plan to achieve its goal based on its current preferences*?

3. AN EXTENDED BDI AGENT MODEL

The example above illustrates the scenario we address in this paper. Our goal is to build software agents that are able to: (i) represent the relationship between plans and (soft-)goals; (ii) represent preferences over softgoals; and (iii) use this information to select plans at runtime. Moreover, we aim to provide a solution by which, with little information about the domain, a software agent's source code can almost completely be generated automatically, requiring no expert knowledge of agent technology. As discussed in the introduction, we propose a model-driven approach in order to achieve our goal, which consists of the following parts:

- an agent representation meta-model, which extends the BDI model, and is inspired by the Tropos meta-model (presented in this section);
- algorithms to process models based on our meta-model and to select plans (Section 4); and
- a means of automatically transforming a model of an agent designed according to our meta-model into source code (Section 5).

Here, we present our meta-model using a top-down approach. We first introduce its top-level concepts, such as an agent, which is composed of lower-level concepts. For space reasons, the concepts of belief and goal are not detailed as they have the same meaning as in the traditional BDI model.

As specified in Definition 1, an agent is an entity able to perform plans in order to achieve its goals, and such plans are selected according to the agent's softgoals and its preferences over them. For example, agent $BobA$ has the goal $GetFunding$ and the plans $Plan\ A$, $Plan\ B$, and $Plan\ C$.

Definition 1 (Agent). *An agent is a tuple*

$$\langle \mathcal{B}, \mathcal{G}, \mathcal{SG}, \mathcal{P}, Pref \rangle$$

where $\mathcal{B}$ is a set of beliefs, $\mathcal{G}$ is a set of goals, $\mathcal{SG}$ is a set of softgoals, $\mathcal{P}$ is a set of plans, and $Pref$ is a preference function.

Agent $BobA$ also has other objectives in addition to the goal of $GetFunding$, typically long-term, which impact on the choice for one plan or another. We name these objectives *softgoals*, a term already adopted in the Tropos meta-model [3]. This concept is also used in the decision making literature, and is referred to as *values* [12], being defined as "*principles used for evaluation. People use them to evaluate the actual or potential consequences of action and inaction, of proposed alternatives, and of decisions.*" Agent $BobA$ has three softgoals, namely $MinTime$, $MinWork$, and $SaveMoney$. A softgoal is defined as below.

Definition 2 (Softgoal). *An agent softgoal $sg \in \mathcal{SG}$ is a broad agent objective, which is not achieved by a plan but is more or less satisfied due to the effect produced by agent actions that are part of plans.*

As stated above, an agent has a set of plans, where a plan is primarily characterised by the goals it can achieve and a body, which consists of a set of actions to be executed. When actions are performed, they produce an effect, for instance completing forms demands effort, which impacts on a softgoal. Each plan may contribute (positively or negatively) to a softgoal, and is thus characterised by a set of contributions to softgoals. We represent the values of contributions as a number between 0 and 1, inclusive, being the worst and best contributions, respectively. However, this contribution may be uncertain. For example, the department typically (with 90% probability) processes requests in one day (contribution value = 1.0), but when it is overloaded with such requests (with 10% probability), it may take up to three weeks to be processed

(contribution value = 0.6), which is acceptable when compared to the two-month average processing time of national agencies. In this way, each plan may have different contributions with different probabilities for each softgoal. Definitions 3 and 4 formally specify plans and contributions respectively.

Definition 3 (Plan). *A plan $p \in \mathcal{P}$ is a tuple*

$$\langle \mathcal{G}', \mathcal{C}, \mathcal{D}, Body \rangle$$

where $\mathcal{G}' \subset \mathcal{G}$ is a subset of goals that the plan can achieve, $\mathcal{C}$ is a set of contributions, $\mathcal{D}$ is a set of dependencies and $Body$ is a set of actions performed when the plan is executed.

Definition 4 (Contribution). *A contribution $c \in \mathcal{C}$ is a tuple*

$$\langle sg, prob, val \rangle$$

where $sg \in \mathcal{SG}$ is a softgoal, and $prob \in [0, 1]$ is the probability of a plan promote the contribution value $val \in [0, 1]$ with respect to sg, 0 and 1 being the lowest and highest possible contributions, respectively. Moreover, $\sum_{c_i \in \mathcal{C} | c_i[sg]=sg} c_i[prob] = 1$.

In some situations, a plan action consists of achieving a particular goal, e.g. while executing $Plan\ A$, agent $BobA$ must achieve the goal *Elaborate Request*, and this leads to a dependency between a plan and a goal. A plan may require that a goal or a set of goals must be achieved during its execution (*And Plan-Goal Dependency*), or one goal of a set must be achieved (*Or Plan-Goal Dependency*). In the latter case, the goal to be achieved is chosen with a given probability. A plan is thus associated with a dependency $\mathcal{D} = AndDep \cup OrDep$, where $AndDep$ and $OrDep$ are plan-goal dependencies, defined next.

Definition 5 (And Plan-Goal Dependency). *An and goal-plan dependency, $andDep \in AndDep$, is a set of goals $g \in \mathcal{G}$, on which a plan depends.*

Definition 6 (Or Plan-Goal Dependency). *An or goal-plan dependency $orDep \in OrDep$ is a set of dependencies, such that each $orDep \in OrDep$ is a pair $\langle g, prob \rangle$, where $g \in \mathcal{G}$ is a goal on which a plan may depend, and $prob \in [0, 1]$ is the probability of the plan need this goal to be achieved during its execution. Moreover, $\sum_{orDep_i \in OrDep} orDep_i[prob] = 1$.*

Finally, agent $BobA$ must take Bob's preferences into account to make a choice. We represent preferences quantitatively, as specified below.

Definition 7 (Preference). *$Pref : \mathcal{SG} \rightarrow [0, 1]$ is a function that maps a softgoal $sg \in \mathcal{SG}$ to a value indicating the preferences for softgoals. A preference is a value $pref_{val} \in [0, 1]$ that indicates the importance of a softgoal $sg \in \mathcal{SG}$, 0 and 1 being the lowest and highest preference, respectively. Moreover, $\sum_{sg \in \mathcal{SG}} Pref(sg) = 1$.*

Such preferences express the *trade-off between different softgoals*. For instance, suppose Bob needs funding for an important conference, he wants to receive a quick response, and does not care too much about the effort it takes. However, as his own funds are limited, he is somewhat concerned with saving money. Preferences for softgoals that reflect this situation may be $MinTime = 0.60$, $MinWork = 0.1$, and $SaveMoney = 0.30$.

The definition of preferences above completes the description of our meta-model. We next describe how we use such concepts to select plans.

4. PROCESSING AGENT MODELS

An instance of our meta-model specifies a set of agents with their associated goals, plans, and other components, which are associated with a specific domain. Now, it is common for such agents is to select plans to achieve goals at runtime, trying to optimise satisfaction of softgoals (detailed in Section 4.1). In addition, as plans may depend on achieving other goals, the contribution of these plans may be derived from the plans executed to achieve those goals. In Section 4.2, we detail how we perform this second task based on our meta-model.

4.1 Selecting Plans

As discussed in the introduction, we aim to support the development of agents with the ability to choose plans at runtime but with little information provided at design time. In order to achieve this, our approach includes an algorithm as part of the agents we generate based on a design model, which chooses a plan according to the current agent softgoals and preferences, and a set of plans to achieve a goal. Our algorithm is based on MAUT [13].

Each plan has a different possible contribution to a softgoal, which depends on what happens when executing the plan. Each plan, with respect to each softgoal, is associated with an **expected contribution**, which is a weighted average of all possible contribution values according to their probability. So, the expected contribution of a plan p with respect to softgoal sg is given by

$$EC(p, sg) = \sum_{c_i \in \mathcal{C} | c_i[sg]=sg} c_i[prob] \times c_i[val]$$

where $\mathcal{C}$ is the set of contributions of p to sg. For example, if there are two plans that achieve the goal *Elaborate Request*, the first with contributions $\langle MinWork, 0.8, 0.3 \rangle$ and $\langle MinWork, 0.2, 0.8 \rangle$, and the second with contribution $\langle MinWork, 1.0, 0.6 \rangle$, their respective expected contributions with respect to $MinWork$ are 0.4 and 0.6. Moreover, when choosing a plan to achieve a goal, the agent preferences should also be taken into account. Such preferences establish a trade-off between softgoals. Thus, the **plan utility** of a plan p for an agent with softgoals $\mathcal{SG}$ and preferences $Pref$ is given by

$$PU(p, Pref, SG) = \sum_{sg \in \mathcal{SG}} Pref(sg) \times EC(p, sg)$$

where $Pref(sg)$ is the preference for a softgoal sg.

Based on the plan utility, we select the plan with the highest utility of a set of possible plans to achieve a goal. This approach to selecting plans is implemented by Algorithm 1; the algorithm has linear complexity, and is simple but effective, as discussed in Section 6, which evaluates our approach.

Note that more than one plan may have the same plan utility, and in this case a plan is selected randomly from those with the maximum utility. In Algorithm 1, we select the first processed plan.

4.2 Deriving Plan Contributions

Our agent meta-model allows the modelling of domain-specific information with respect to the satisfaction of softgoals, i.e. the contribution of each plan to each softgoal. However, as the execution of certain plans may depend on achieving other goals, specifying contributions for each plan may be redundant. For example, Plan A of our example is composed of a set of actions of which some consist of the achievement of the goals: *Elaborate Request*, *Submit Request*, and so on. Therefore, the contribution of the Plan A may be computed based on the combination of the contributions of the plans to achieve the goals that Plan A depends

Algorithm 1: $SelectPlan(\mathcal{SG}, Pref, \mathcal{P})$

Input: $\mathcal{SG}$: set of agent softgoals, $Pref$: map of agent preferences; $\mathcal{P}$: set of plans that can achieve a goal
Output: $selectedP$: plan that best satisfies agent preferences

```
1  selectedP ← null;
2  maxContrib ← null;
3  foreach p ∈ P do
4      contrib ← 0;
5      foreach sg ∈ SG do
6          possibleContribs ← Contribution(p,sg);
7          expectedContrib ← 0;
8          foreach ⟨prob, value⟩ ∈ possibleContribs do
9              expectedContrib ← expectedContrib + prob × value;
10         contrib ← contrib + Pref(sg) × expectedContrib;
11         if selectedP = null ∨ maxContrib < contrib then
12             selectedP ← p;
13             maxContrib ← contrib;
14 return selectedPlan;
```

on, if there are no contributions with respect to a particular softgoal specified for Plan A.

As stated above, we have two types of dependency: **and** and **or**, and they are treated differently. However, both dependencies share a common feature: since plans depend on achieving goals, and a goal may be achieved by different plans, we must first estimate the satisfaction of softgoals when achieving a goal g, considering the plans that may be executed to achieved it. As discussed above, each plan can be associated with an expected contribution $EC(p, sg)$. Given that more than one plan may achieve g, the contribution of g with respect to sg is the mean of the expected contributions of each plan that may achieve it, with probability 1.0. We adopt the mean because choosing a plan also depends on preferences, which are not fixed, and vary at runtime. For example, the *expected contribution* associated with the goal *Elaborate Request* — discussed in the previous section — with respect to the softgoal $MinWork$ is $\langle MinWork, 1.0, 0.5\rangle$.

Now, we must combine the contributions of the goals on which a plan depends, so that we can derive its contributions. An **and** plan-goal dependency specifies a **set of goals** that must be achieved during the execution of plan p. Therefore, the contributions of p with respect to a softgoal sg is the **sum** of all expected contributions of goals for which p has an **and** dependency, normalised to $[0, 1]$, with probability 1.0. An **or** plan-goal dependency specifies a set of goals of which **one** must be achieved, and this dependency also specifies the probability of choosing each particular goal to be achieved. Therefore, the contributions of p with respect to a softgoal sg, in this case, are the expected contributions of goals on which p has an **or** dependency, with the probability specified in this dependency relationship.

5. MODEL-TO-CODE TRANSFORMATION

Thus far, we have presented our meta-model, with its concepts of softgoal, contribution and preferences, and how to use them in the plan selection process. We now focus on how to transform design models based on our meta-model into implemented agents. Our approach provides a meta-model together with production rules that capture the knowledge to transform meta-model concepts into code, running over our provided infrastructure. So, given a meta-model instance (i.e. a model, of a particular application), we use a code generator to perform a model-to-text transformation. During this process we perform additional computations, such as calculating plan contributions based on dependencies. The implementation of plan bodies is left to application developers.

However, since there is a gap between the concepts of our meta-model and abstractions provided by existing agent platforms, we first develop or extend a platform in which agents are executed to make the transformation simpler. There are many existing BDI platforms, such as JACK [10], Jadex [18], Jason [2], and BDI4JADE[1] [16]. While extending most of these platforms requires changing how XML or similar files are processed, in BDI4JADE agents are developed solely with the constructions of a general purpose programming language (Java), making it is easier to extend, so we adopt this particular platform in our work.

Production Rule	Transformation
Agent	**name** extends UtilityBasedBDIAgent init() `ForEach` goal → **AddGoal** add softgoals (in the Softgoals file) initPreferences() `ForEach` plan → **AddPlan** initPreferences() `ForEach` softgoal → **SetPreference**
	`ForEach` plan → **Plan**
	`ForEach` goal → **Goal**
	Softgoals (generate Softgoals file)
AddPlan	Add plan to library
AddGoal	Add goal to agent
SetPreference	Set preference belief value associated with the softgoal
Plan	**PlanDefinition**
	PlanBody
PlanDefinition	**name** extends SimplePlan Constructor() get contribution metadata `ForEach` contribution → **SGContribution**
SGContribution	Create softgoal contribution array `ForEach` contribution → **Contribution** Add contributions to contribution metadata
Contribution	Add plan contribution (pair of probability and value)

Table 1: Production Rules.

We illustrate how we implemented our approach by extending BDI4JADE, in Figure 1 — our extensions are in white, while existing BDI4JADE components are in grey. The main extensions are as follows. First, BDI agents have a set of **softgoals**. Second, as BDI4JADE plans can have **metadata**, we store as their default metadata **plan contributions and dependencies**. Third, we include a **plan selection strategy** (an extension of BDI4JADE) according to our plan selection algorithm based on MAUT. Finally, **utility-based BDI agent** is an extension of a BDI agent, whose plan selection strategy is set to our algorithm and which also has a **belief named *SoftgoalPreferences*** that stores preferences over softgoals.

Given that this infrastructure provides a means of implementing agents with our approach, we want to transform agents designed based on our meta-model to an implemented agent. This transformation is performed through the use of production rules, which transform instantiated concepts of our meta-models to code pieces, or to other production rules that will eventually produce code. Due to space restrictions, we are not able to present all production rules needed for our approach, but we present some of them as illustration in Table 1. Rules in italics in Table 1 have their transformation omitted. The production rule column indicates the rule name which, when applied, is transformed into the content presented in the transformation column. Each rule is applied in the context of a concept of our meta-model, and consequently it may manipulate the concept properties, such as iterating using `ForEach`. The transformation process begins by applying the **Agent** rule to all agents instantiated in the model.

These production rules were implemented using the Xpand[2] tem-

[1] `http://www.inf.ufrgs.br/prosoft/bdi4jade/`
[2] `http://www.eclipse.org/modeling/m2t/`

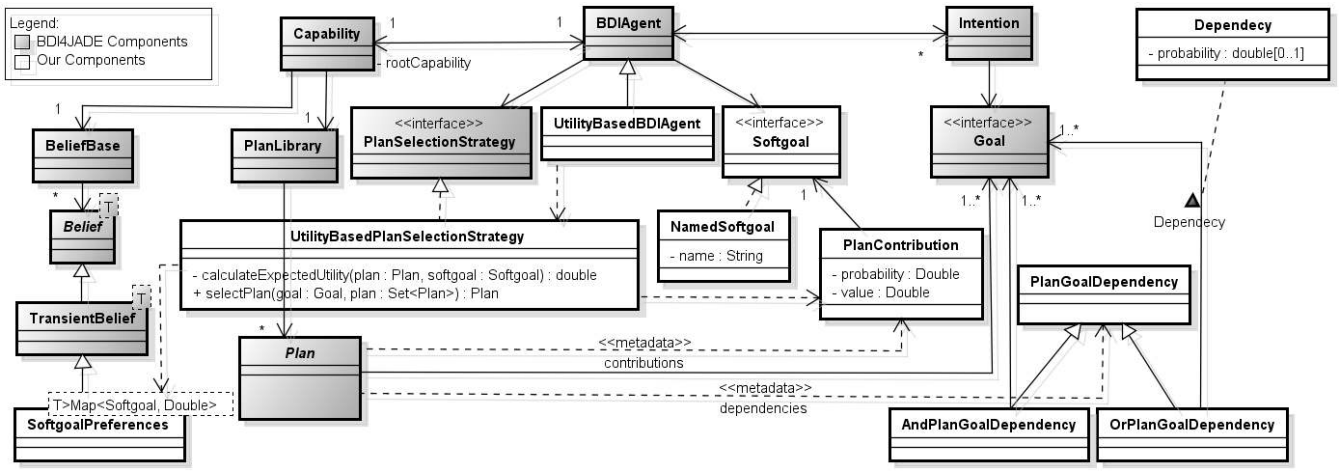

Figure 1: Extended BDI Agent Meta-model in the BDI4JADE.

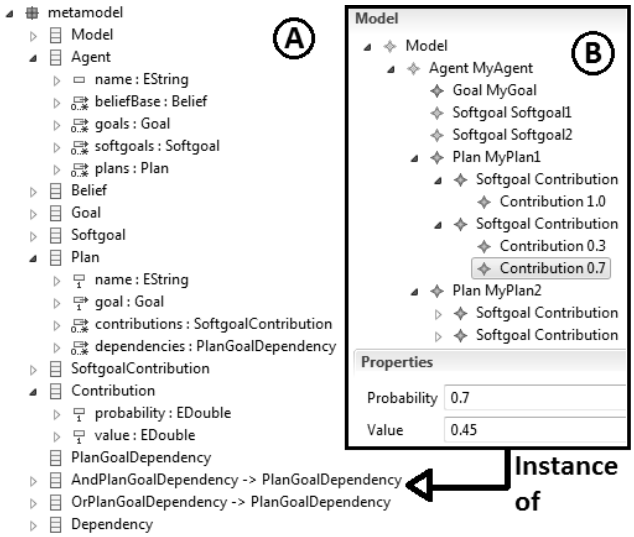

Figure 2: Meta-model and Model.

```
«DEFINE softgoalsClass FOR Agent»
«FILE name + "/" + name + "Softgoals.java"»
package «this.name»;

import bdi4jade.softgoal.*;

public class «this.name»Softgoals {
    // Softgoals
    «EXPAND softgoalDefiniton FOREACH this.softgoals»
    public static final Softgoal ALL_SOFTGOALS[] = {
        «EXPAND softgoalArray FOREACH this.softgoals»
    };
    private «this.name»Softgoals() { }
}
«ENDFILE»
«ENDDEFINE»

«DEFINE softgoalDefiniton FOR Softgoal»
public static final Softgoal «this.name» =
     new NamedSoftgoal("«this.name»");
«ENDDEFINE»

«DEFINE softgoalArray FOR Softgoal»«this.name»,«ENDDEFINE»
```

Figure 3: Code Template Example.

plate language, part of the Model to Text (M2T) project of Eclipse. We first implemented our meta-model using M2T as specified in Section 3 — see Figure 2(A). Then, the rules were implemented as templates as presented in Figure 3, which gives an example of a part of the template that generates the agent softgoals, which are declared in a class. This template corresponds to the **Softgoals** rule.

In order to generate agents based on our implementation, an instance of the meta-model should be created, as shown in Figure 2(B). Then, a workflow must be executed, which first processes plan-goal dependencies to derive plan contributions and then executes production rules generating source code.

6. EVALUATION

Now that we have detailed how agents with the ability to select plans can be automatically generated based on an instance of our meta-model, we evaluate our approach with an empirical experiment. Even though the use of preferences in plan selection [21] (discussed in the related work section) has previously been considered, it does not address probabilities, and expresses preferences in a different way. However, its representation of expressing preferences can be mapped to ours without probability, so if this approach [21] were adopted for our evaluation scenario, it would always select the same plan. In consequence, we compare our approach to random plan selection, with the settings described in Section 6.1. We then present our results in Section 6.2, and a discussion in Section 6.3.

6.1 Experiment Settings

Our experiment consists of a simulation to compare the accumulated satisfaction of an agent after executing a plan to achieve a goal, when using our approach to select the plan and when selecting it randomly from a set of possible plans. The satisfaction of an agent is calculated based on how softgoals are satisfied. As the outcome of executing a plan is uncertain, the satisfaction of a plan that has the best (expected) plan utility is not necessarily that plan that promotes that highest satisfaction.

?project=xpand

	Bicycle		Bus		Car		Moto	
Softgoal	Prob	Val	Prob	Val	Prob	Val	Prob	Val
Safety	0.05	0.00	0.10	0.00	0.15	0.00	0.30	0.00
	0.95	1.00	0.90	1.00	0.85	1.00	0.70	1.00
Security	0.20	0.00	0.18	0.00	0.15	0.00	0.20	0.00
	0.80	1.00	0.82	1.00	0.85	1.00	0.80	1.00
Performance	0.05	0.00	0.10	0.00	0.15	0.00	0.30	0.00
	0.475	0.00	0.45	0.33	0.425	0.61	0.35	0.77
	0.475	0.04	0.45	0.45	0.425	0.77	0.35	0.88
Cost	0.05	0.00	1.00	0.65	0.15	0.00	0.30	0.00
	0.95	0.90			0.85	0.10	0.70	0.45
Comfort	1.00	0.20	1.00	0.40	1.00	0.80	1.00	0.50

Table 2: Plan Contributions to Softgoals.

The simulation scenario is an agent whose goal is to *go to work from its home*. This agent has four possible plans: (i) go by bicycle; (ii) take a bus; (iii) drive a car; and drive a motorcycle. In addition, this agent has five softgoals, detailed below.

Maximise Safety Maximising safety in this context is associated with avoiding the possibility of crashing. While going to work, the agent may crash, and the probability of crashing depends on the type of transportation adopted. For instance, the probability of crashing with a motorcycle is much higher than crashing with a car.

Maximise Security Maximising security in this context is associated with avoiding the possibility of being robbed. For example, if an agent stops at a traffic light on a motorcycle, a robber may steal its backpack, or if on a bus, a robber may enter and steal from all passengers.

Maximise Performance Maximising performance means spending less time to go to work. Motorcycles can ride at a high speed and pass through traffic jams, so they are faster than other transportation types. If the agent crashes while going to work, it takes the worst possible time to arrive at work.

Minimise Cost Minimising cost means spending less money on transportation. Using a bicycle is very cheap, as maintaining it costs very little, while maintaining a car is very expensive. However, if an agent crashes while going to work, it will spend a lot of money to repair its bicycle, car or motorcycle (which is not the case with a bus).

Maximise Comfort Maximising comfort means travelling in a more comfortable way and with less effort. For example, a car is more comfortable than a bus.

Using the rationale described above, each of the plans has different contributions (probability and value) to each softgoal, as detailed in Table 2. For example, the probability of a motorcycle crash is 0.3. Therefore, if the motorcycle plan is chosen, the contribution of this plan with respect to safety is 0.0 (if the agent crashes) with probability 0.3, and 1.0 (if the agent does not crash) with probability 0.7. This is also the case with the probability of being robbed and the security softgoal. In addition, crashing also promotes the worst cost and performance. But if there is no crash, the value associated with the cost softgoal is specific to each transportation type, while that associated with the performance softgoal corresponds to the minimum and maximum time taken to get to the destination on a scale from 0 to the maximum possible time that can be taken. Finally, the value associated with the comfort softgoal corresponds to how comfortable each transportation type is.

Our experiment consists of running a number of iterations in which we perform the following steps.

Step 1 Randomly generate preferences for each softgoal.

Plan Selector	M	SD	Min	Max	Cum Sat
Bicycle	*0.57*	0.16	0.004	**0.97**	*2829.28*
Bus	0.63	*0.14*	0.05	0.95	3136.73
Car	0.64	0.18	0.002	0.97	3197.90
Moto	0.58	**0.24**	*0.0001*	0.96	2904.73
Random	0.60	0.19	0.0002	0.95	3001.09
Utility-based	**0.66**	0.19	*0.0001*	**0.97**	**3301.38**

Table 3: Satisfaction by Plan Selector (n = 5000).

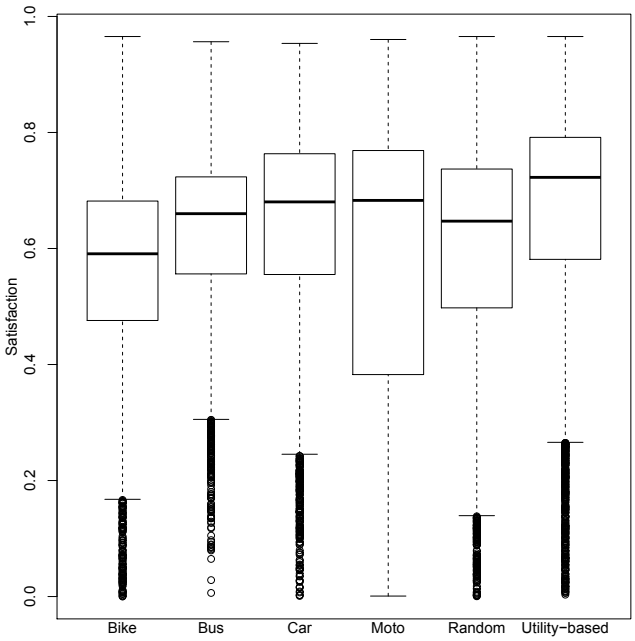

Figure 4: Analysis of Satisfaction by Plan Selector.

Step 2 Randomly instantiate a scenario for each plan, according to the given probability of events (crashing, being robbed, time taken, etc.).

Step 3 Compute the satisfaction for each such scenario.

Step 4 Select a plan for each plan selector (our algorithm and randomly).

Step 5 Store the satisfaction of the scenario associated with the selected plan.

6.2 Results and Analysis

In our experiment, we ran 5000 iterations of the steps described above, each of which takes less than 1 second to run. As result, we compared the average satisfaction and the accumulated satisfaction of all iterations for each plan selector (random and utility-based). Moreover, we also analysed constant plan selectors: those that always choose the same plan. The average satisfaction obtained, the standard deviation and minimum and maximum values, and accumulated satisfaction are detailed in Table 3, and we present the box plot of the satisfaction by plan selector in Figure 4. In Table 3, the highest values are in bold and the lowest values are in italics.

As can be seen in both Table 3 and Figure 4, the plan selector with the best results is the utility-based plan selector, while the bicycle plan selector has the worst results. Therefore, even with an uncertain outcome when selecting a plan, our approach manages to achieve the best average satisfaction for the agent. However, this is not the case for every individual iteration, since the utility-based plan selector chooses the plan with the best *expected* value, but an undesired event, such as a crash or being robbed, could cause other

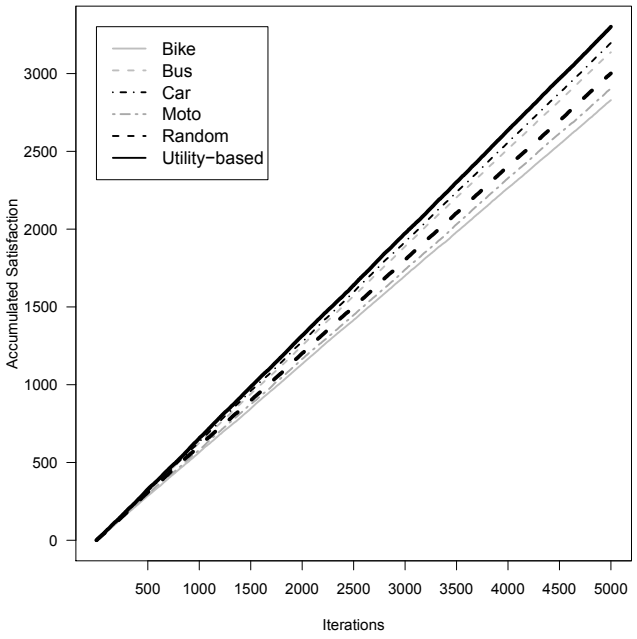

Figure 5: Accumulated Satisfaction by Plan Selector.

plans to be more successful. This uncertainty is clearly seen in the results of the the motorcycle plan selector, which is associated with high standard deviation (0.24) that can also be observed in Figure 4. As a consequence of choosing the motorcycle, the agent may get very satisfied (very good performance and good costs) or very unsatisfied (if the agent crashes). Even though using a car achieves the highest average satisfaction, the utility-based plan selector chooses the plan that best fits the agent preferences — an agent may want to take the risk of crashing if this increases its chances of arriving at its destination more quickly. Therefore, our plan selector selects a different plan for a different set of preferences. The number of times that each plan was chosen is as follows: (i) bicycle = 307; (ii) Motorcycle = 1026; (iii) Car = 2903; and (iv) Bus = 764.

In order to consider the impact of using a plan selector over time, we also show in Figure 5 the accumulated satisfaction obtained after running 5000 iterations. The difference increases over time, but in the very first iterations this difference is small, due to the uncertainty of the scenario that arises in the selection of a plan.

After testing for normality, a one-way ANOVA was used to test for preference differences among satisfaction of each plan selector. Satisfaction of plan selectors differed significantly across the six selectors, $F(5, 29994) = 190.7$, $p \ll .05$. Post-hoc Tukey's HSD tests showed that all comparisons were significantly different at .05 level of significance.

6.3 Discussion

As discussed above, our plan selector significantly increases an agent's satisfaction in comparison to other plan selectors. The cases where our plan selector had worse results are associated with the uncertainty of events that may arise while executing a plan, such as the occurrence of a crash.

Our experiment allowed us to identify a limitation of our approach: the representation of dependent probabilities. For almost all plans, the expected outcome depends on the occurrence of a crash, and its probability is used to set up the contributions of the safety, cost and performance softgoals. However, this dependency is not captured in our meta-model, which may facilitate the domain modelling. We also assume that each plan given as input to our plan selection algorithm has the same set of contributions, i.e. contributions associated with the same set of softgoals. Therefore, we currently do not deal with unknown information.

Another limitation of our approach is the use of quantitative values for preferences and contributions. This is a widely known issue in the context of preference reasoning, and existing preference elicitation algorithms can be adopted to obtain such values. Although there is no restriction to set up and change plan contributions at runtime, we assume that they are set up at design time, so that the effort of specifying these values is performed just once.

7. RELATED WORK

In this section, we discuss work related to our approach, divided into three categories. First, we discuss work to adopt a model-driven approach to develop MASs. Second, we present approaches that use the concepts also used in our own work. Finally, we detail and compare our approach to work on preference-based plan selection. The adoption of Model-driven Development (MDD) for building MASs has been investigated by the Agent-Oriented Software Engineering (AOSE) community in recent years [5, 6, 8]. MDD considers models as first class citizens in software development, and aims to automatically generate code for software applications by transforming and composing models. Note that this idea is only effective when the level of abstraction used in models is higher than that used in the source code. On some occasions, the focus is to achieve platform independence, which is the case with Model-driven Architecture (MDA),[3] which separates the business and application logic from the underlying platform technology. According to MDA, a platform-independent model (PIM) is translated to platform-specific models (PSMs), based on a Computation Independent Model (CIM). Although model-driven approaches for developing MASs have made substantial progress in achieving platform independence as they are based on MDA, they have made little progress in generating source code that goes beyond skeletons of key MAS concepts [15]. Our approach, on the other hand, produces agents that are provided with the ability to select plans.

With respect to the concepts used in our approach, as discussed previously, the components of our meta-model are inspired by the Tropos agent methodology [3]. Many agent methodologies based on the BDI model have been proposed [9], but they typically represent solely the BDI concepts, without detailing properly their relationships. Tropos, derived from the i* [22] framework (used to model functional and non-functional requirements), differs from most other methodologies by explicitly modelling, for example, the relationship between plans and goals and goal decomposition. Furthermore, it includes the concept of softgoals. In addition to adopting the concept of softgoals, our approach considers different dependency relationships (to be able to derive plan contributions), preferences over softgoals and uncertain plan contributions. Moreover, we also provide a means of using this information at runtime to select plans. Our approach does not address goal decomposition, because it is neither used in the plan selection process nor to generate agent code. At the implementation level, the Jadex platform [18] implements the concept of meta-goal, which is a goal to select a plan and is achieved by meta-plans. This is equivalent to the plan selection strategy of BDI4JADE, and is just an extension point of the platform, but the plan selection itself should be implemented for specific applications.

Finally, a recent approach has proposed using preferences for se-

[3] http://www.omg.org/mda/

lecting plans [21]. This approach extended a language for preference-based planning [1], to use it with agents that have a pre-determined plan library, as in our case. This approach expresses preferences over individual plan characteristics, and not over a broader general goal, our softgoals, which may affect reuse negatively. If we expressed the case of our empirical experiment with this approach, the plans would be characterised by the attributes crashed, robbed, time and so on, and preferences would be over possible values of these attributes. Therefore, by expressing preferences as plan contributions, we abstract a very specific plan description, which requires domain knowledge. Moreover, the preferences of our approach express trade-offs between softgoals, which cannot be modelled with the Visser et al.'s approach, as is the case for the uncertainty considered in our approach. By using the semantics of the extended language proposed by Visser et al. and translating the plan contributions of our simulation to their approach, the car and motorcycle plans would be (equally) the most preferred plans and thus always selected,[4] which are the plans that have the first and third best average satisfaction, respectively — but our approach has an even better result. The advantage of this approach [21] is that our notion of contributions can be derived from their preferences so that, depending on the information available about the domain being modelled, both approaches may be used in a complementary way.

8. CONCLUSION

Model-driven Development is a promising approach for building applications based on high-level models that, with transformations and compositions, may eventually produce an implemented system, and successful industry projects provide evidence of the potential of this direction [11]. It has also been investigated in the context of multi-agent system (MAS), but current work is still premature.

In this paper, we proposed a model-driven approach to develop BDI agents able to select plans based on softgoals and preferences. Our approach consists of a meta-model that specifies concepts that allow representation of the information needed for the plan selection process, such as plans, their dependencies, and their contribution to softgoals. We also proposed a simple but effective algorithm based on MAUT that selects a plan based on plan contributions and agent preferences, considering an uncertain outcome of the plan executing. Finally, we described how to transform a model of an agent designed with our meta-model into source code. The effectiveness of our approach was shown with an empirical evaluation.

As future work, we will extend this approach to represent dependent probabilities in plan contributions, and also to use languages and algorithms [17] to represent and reason about qualitative preferences. Moreover, we aim to address other issues of BDI agents, such as goal generation and selection, and incorporate them into our model-driven approach.

9. ACKNOWLEDGMENTS

Work supported by FAPERGS PRONEX 10/0049-7.

10. REFERENCES

[1] M. Bienvenu, C. Fritz, and S. A. McIlraith. Planning with qualitative temporal preferences. In *KR*, pages 134–144. AAAI Press, 2006.

[2] R. H. Bordini, M. Wooldridge, and J. F. Hübner. *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.

[4]This approach requires choosing a parameter (*k-estimate*) of the resource usage, and we choose average values taking probabilities into account.

[3] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[4] A. Dasgupta and A. K. Ghose. Implementing reactive bdi agents with user-given constraints and objectives. *Int. J. Agent-Oriented Softw. Eng.*, 4(2):141–154, 2010.

[5] K. Fischer, C. Hahn, and C. Madrigal-Mora. Agent-oriented software engineering: a model-driven approach. *IJAOSE*, 1(3/4):334–369, Dec 2007.

[6] I. García-Magariño, J. Gómez-Sanz, and R. Fuentes-Fernández. Model transformations for improving multi-agent system development in ingenias. In *AOSE'10*, pages 51–65. Springer, 2011.

[7] P. Giorgini, J. Mylopoulos, and R. Sebastiani. Goal-oriented requirements analysis and reasoning in the tropos methodology. *EAAI*, 18(2):159–171, 2005.

[8] C. Hahn, C. Madrigal-Mora, and K. Fischer. A platform-independent metamodel for multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 18(2):239–266, 2009.

[9] B. Henderson-Sellers and P. Giorgini. *Agent-oriented Methodologies*. Idea Group Publishing, 2005.

[10] N. Howden, R. Rönnquista, A. Hodgson, and A. Lucas. Jack intelligent agents™: Summary of an agent infrastructure. In *Autonomous Agents*, 2001.

[11] J. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristoffersen. Empirical assessment of mde in industry. In *ICSE '11*, pages 471–480. ACM, 2011.

[12] R. L. Keeney. *Value-focused thinking – A Path to Creative Decisionmaking*. Harvard University, 1944.

[13] R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. John Wiley & Sons, Inc, New York, 1976.

[14] F. Meneguzzi and L. De Silva. Planning in bdi agents: a survey of the integration of planning algorithms and agent reasoning. *The Knowledge Engineering Review*, FirstView:1–44, 9 2013.

[15] I. Nunes, D. Cowan, E. Cirilo, and C. Lucena. A case for new directions in agent-oriented software engineering. In *AOSE'10*, pages 37–61. Springer, 2011.

[16] I. Nunes, C. Lucena, and M. Luck. Bdi4jade: a bdi layer on top of jade. In *ProMAS 2011*, pages 88–103, Taiwan, 2011.

[17] I. Nunes, S. Miles, M. Luck, and C. Lucena. User-centric principles in automated decision making. In *SBIA 2012*, volume 7589 of *LNCS*, pages 42–51. Springer-Verlag, 2012.

[18] A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: A bdi reasoning engine. In *Multi-Agent Programming*, pages 149–174. Springer, 9 2005.

[19] A. S. Rao and M. P. Georgeff. BDI-agents: from theory to practice. In *First Intl. Conf. on Multiagent Systems*, 1995.

[20] T. Stahl, M. Voelter, and K. Czarnecki. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006.

[21] S. Visser, J. Thangarajah, and J. Harland. Reasoning about preferences in intelligent agent systems. In *IJCAI'11*, pages 426–431. AAAI Press, 2011.

[22] E. S. K. Yu. Towards modeling and reasoning support for early-phase requirements engineering. In *RE '97*, pages 226–235. IEEE Computer Society, 1997.