BDI Agents with Objectives and Preferences

Aniruddha Dasgupta and Aditya K. Ghose

Decision Systems Lab
School of Computer Science and Software Engineering
University of Wollongong,
Wollongong, NSW 2522, Australia
{ad844,aditya}@uow.edu.au

Abstract. For many applications there is the need to handle user preferences and customize agents according to the user's specific needs. It is convenient to let the user provide elaborate specification consisting of constraints, preferences and objectives. Then, let the agent system make decisions about its actions by taking into account changes in the surrounding environment as well as the user preferences that come in real-time. In this paper we describe an agent programming language where we incorporate constraints, objectives and preferences into the BDI framework. Our work especially focuses on the use of soft constraints in an agent environment where we give a quantitative dimension to this agent deliberation process by apply c-semiring based techniques to determine the preferred solution.

1 Introduction

In the *multi-agent systems* (MAS) community, software agents are conceived as autonomous computational entities situated in some environments which they can sense and act upon in a dynamic (reactive and/or proactive) way according to the environment's changes and their design objectives [25]. Each agent is given the mandate to achieve defined goals. To do this, it autonomously selects appropriate actions, depending on the prevailing conditions in the environment, based on its own capabilities and means until it succeeds, fails, needs decisions or new instructions or is stopped by its owner. Thus decision agents can be designed to provide interactive decision aids for end-users by eliciting their preferences and then recommending matching products.

BDI [18] agent-oriented systems are flexible and responsive to the environment, and well suited for complex applications with real-time reasoning and control requirements [20]. However, not much work has been done regarding the practical implementation of BDI languages that incorporate user preferences into the BDI framework.

In this paper, we develop a traditional BDI-style agent programming language that has built-in decision making strategies based on user preferences. These preferences could be modeled as either hard constraints (constraints that must be satisfied with explicit objectives like maximise or minimize *cost*) or soft constraints (constraints that the user would *like* to satisfy). We call this language BAOP (BDI Agent with Objectives and Preferences). Our work focuses on practical means-ends reasoning which deals with what actions to perform and how to perform the actions. We implement BAOP by extending CASO [7] and incorporating a mechanism by which user preferences can be added into the system.

The contributions of this paper are fivefold. Firstly, a BDI language framework is developed where user preferences can be handled by the system. Secondly, techniques are described whereby a particular behavior can be selected according to the preferences. Thirdly, the language framework is modified whereby we parameterize basic actions. Fourthly, formal semantics of the language is described and fifthly, we describe a method by which preferences and objectives can be integrated.

The remainder of this article is organized as follows. Section 2 gives a brief background on related work on BDI languages and constraints, section 3 describes the syntax of the language section 4 gives an overview of its operational semantics. Experimental results and concluding remarks are provided in the last sections.

2 Background

2.1 BDI Languages

One of the most popular and successful framework for Agent technology is that of Rao and Georgeff [18], in which the notions of *Belief*, *Desire* and *Intention* are central, and hence are often referred to as BDI agents. Beliefs represent the agent's current knowledge about the world, including information about the current state of the environment inferred from perception devices and messages from other agents, as well as internal information. Desires represent a state which the agent is trying to achieve. Intentions are the chosen means to achieve the agent's desires, and are generally implemented as plans and post-conditions. As in general an agent may have multiple desires, an agent can have a number of intentions active at any one time. These intentions may be thought of as running concurrently, with one chosen intention active at any one time.

AgentSpeak(L) [17] is one of the most influential abstract languages based on the BDI architecture. It is an agent framework/language with explicit representations of beliefs and intentions for agents. AgentSpeak(L) is a programming language based on a restricted first-order language with events and actions. Due to its simplicity and elegance, AgentSpeak(L) can be easily extended. There have been several languages based on AgentSpeak(L) and Jason [3] is one of its well-know interpreters.

There are a number of other agent programming languages in the BDI tradition, such as 3APL [10], JACK [5], CASL, [22], Dribble [23] and CAN [24].

2.2 Hard and Soft Constraints

Hard constraints are those which we definitely want to be true. These might relate to the successful assembly of a mechanism. Soft constraint are those we would like to be true - but not at the expense of the others. These might say that a mechanism must follow a given path. There is not point in trying to match every point exactly if this can only be done by breaking the assembly of the links.

Soft constraints model quantitative preferences by generalizing the traditional formalism of hard constraints. In a soft constraint, each assignment to the variables of a constraint is annotated with a level of its desirability, and the desirability of a complete assignment is computed by a combination operator applied to the local preference values. By choosing a specific combination operator and an ordered set of levels of desirability, a specific class of soft constraints can be selected.

A soft constraint may be seen as a constraint where each instantiation of its variables has an associated value from a partially ordered set which can be interpreted as a set of preference values. Combining constraints will then have to take into account such additional values, and thus the formalism has also to provide suitable operations for *combination* (x) and *comparison* (+) of tuples of values and constraints. Semiring-based constraint satisfaction proposed by Bistarelli et.al [2] is a meta-approach for modelling problems with preferences. This framework uses a semiring structure, where the set of semiring specifies the preference associated to each tuple of values. The two semiring operations (+ and x) then model constraint projection and combination respectively.

In semiring-based constraint satisfaction, each tuple in the constraint is marked by a preference level expressing how good the tuple satisfies the constraint. The preference level is taken from a set A equipped with the c-semiring structure (A,+,x,0,1). A is a set of preferences, + is a commutative, associative, idempotent (a+a=a) binary operation on A with the unit element 0 (0+a=a) and the absorbing element 1 (1+a=1), x is a commutative, associative binary operation on A with the unit element 1 (1xa=a) and the absorbing element 0 (0xa=0) and x distributes over +.

The multiplication operation x is used to combine constraints. Let *vars*(c) be a set of variables over which the constraint c is defined, δc be a mapping of all tuples over *vars*(c) to A, i.e., $\delta c(V)$ is a preference of the tuple V in the constraint c, and let $U \downarrow Y$ be a projection of some tuple U to variables Y. Then we can describe a preference of some tuple V by combining preferences of this tuple (its projection) in all the constraints C:

$$p(V) = \mathrm{x}_{c \in C} \delta c(V \downarrow vars(c))$$

To compare the preferences of tuples we need some ordering on A. This ordering can be defined using the additive operation + in the following way: $a \leq b \Longleftrightarrow a + b = b$. If $a \leq b$ then we say that b is better than a. Note that the relation $\leq$ defines a partial ordering on A opposite to the total ordering used in the valued constraint satisfaction.

The *semiring-based constraint satisfaction problem* is defined formally by the c-semiring structure (A,+,x,0,1), the set of variables X, their domains D, and the set of constraints C described via δc. The task is to find an assignment V with the best preference *p(V)*.

2.3 Using Constraints in BDI Languages

Incorporating constraints into BDI languages can be be of great advantage to agent decision making. Some of the advantages of using constraints are:

- Constraints can capture qualitative and quantitative preferences and costs.
- Constraints offer a declarative representation that is easy to understand.
- Constraints are supported by a large set of algorithms, solvers, and tools.

In our earlier work in [6] and [7], we have shown that the concept of using constraints and explicit objectives in a high-level agent specification language like AgentSpeak(L) yields significant advantages in terms of both expressivity and efficiency. This technique applies constraint and objective directed solving on the context section of a BDI agent's plan specification in order to determine an application plan to fire. The new language is called CASO (Constraint AgentSpeak(L) with objectives). We have also defined efficient plan and intention selection techniques with the notion of parametric look-ahead. An implementation of CASO is also described in [7], which provides the user with the flexibility of adding explicit objectives and constraints to achieve final goals. CASO uses a modified version of the Jason interpreter, together with another open-source constraint solver ECLiPSe [1], thereby combining reactive agent programming with constraint solving techniques.

3 BAOP: A Reactive BDI Language

In this section we give an overview of our BDI based language BAOP and describe its syntax. BAOP is a programming language based on AgentSpeak and is implemented using Jason.

Informally, an agent program in BAOP consists of a set of beliefs β which includes a set of constraints, a set of objective functions Θ, a set of user preferences μ, a set of events E, a plan library P, a set of intentions I, an objective store OS and a preference store PS. There are three selection functions S_E, S_P, S_I to select an event, a plan and an intention respectively. There is also a look-ahead parameter n which determines how many steps the agent is going to look ahead before committing to a plan or intention.

Let us now explain each of terms mentioned in the above informal definition,

3.1 Belief Base (β)

β is a Constraint Logic Program(CLP)[13] and not a just a set of simple facts. As we will see later, such an approach which combines the flexibility of logic with the power of search to provide high-level constructs for solving computationally hard problems can help an agent to choose a plan or intention intelligently.

3.2 Set of Plans (*P*)

P is a repository which contains all the available pre-compiled plans for the agent to use. When a triggering event occurs, all the plans triggered by this event that can be executed in the current circumstances are retrieved. Below we define a BAOP plan.

Definition 1. *A BAOP plan p is of the form* $t[\varepsilon]$:$b_1 \wedge b_2 \wedge \cdots \wedge b_n \wedge c_1 \wedge c_2 \wedge \cdots \wedge c_m \leftarrow sg_1, sg_2, \cdots, sg_k$ *where* t *is the trigger;* ε *refers to the effect of the plan; each* b_i *refers to a belief; each* c_i *is an atomic constraint; each* sg *is either an atomic action or a subgoal.*

It should be noted that in the definition of the plan above, an *action* could have *parameters* whose values are instantiated when the agent actually executes the plan. Also, since CLP assumes Horn Clause, the *effets* can be Horn Clauses only as we use the effets together with β as we will see later.

3.3 Set of Objective Functions (*Θ*)

Θ represent objective functions like *maximize(exp)* or *minimize(exp)* where *exp* consists of global variables that are valid throughout the lifetime of the agent. Objectives represent quantitative measure of goals that the agent would like to achieve.

3.4 Set of User Preferences (*μ*)

μ is the set of preferences that the agent would like to achieve. Along with objectives, this is yet another natural way of representing softgoals. The preference is given using semiring values and is of the form $< \varepsilon, v_1 >$ which depicts a preference value v_1 for pursuing the plan. ε denotes the cumulative effect of plan (or plans).

3.5 Set of Events (*E*)

E is the set of events which could be external or internal. Agents talk to the external environment through events. The different types of external events which originate from perception of the agent's environment:

1. Addition and deletion of beliefs (with constraints).
2. Addition and deletion of achievement goals.
3. Addition and deletion of test goals.
4. Addition and deletion of objectives.
5. Addition and deletion of user preferences.

The first three types of events are triggering events (where the context of the plan is matched with relevant plans), while the last two are non-triggering.

Internal events are generated from the agent's own execution of a plan (i.e., as a subgoal in a plan generates an event of the type addition of an achievement goal). An internal event is accompanied with the intention which generated it (as the plan chosen for that event will be pushed on top of that intention).

3.6 Set of Intentions (I)

Intentions are particular courses of actions to which an agent has committed in order to handle certain events. I consists of a set of intentions where each intention is a stack of partially instantiated plans.

3.7 Objective Store(OS))

OS a consistent set of objective functions and is updated in case a new objective comes in as an event. Below we give the formal definition of what it means by augmenting the OS.

Definition 2. *Given an objective store OS and a new objective f, the result of augmenting OS with f, denoted by OS^*_f, is defined as $\gamma(MaxCons(OS \cup f))$ where γ is a choice function and $MaxCons(X)$ is the set of all $x \subseteq X$ such that x is consistent and there exists no x' such that $x \subset x' \subseteq X$ and x' is consistent.*

The new OS is now given by $\gamma(MaxCons(OS \cup \overline{O}) \cap OS)$ where γ is the choice function, and $\overline{O}$ is the negation of the objective O.

Formally a *consistent objective store* is defined as below.

Definition 3. *Objectives O_1 and O_2 are inconsistent if and only if there exists a pair of solutions S_1 and S_2 such that S_1 is preferred over S_2 by O_1 and the reverse holds under O_2.*

3.8 Preference Store (PS)

PS a consistent set of user preferences and is updated in case a new preference comes in as an event. A PS is inconsistent if there exists at least two tuples whose conditions are logically equivalent but whose associated semiring values are different. The machinery we provide ensures such inconsistencies do not occur. The consistency of user preferences in the preference store is maintained by the following logic:

When a new preference comes, it is compared to the set of preferences that are currently in the preference store. If the new user preference tuple is $< \varepsilon, v_1 >$ and if PS contains a tuple with ε'_1 which is logically equivalent to ε_1 then replace the value of ε'_1 with v_1 else insert the new tuple in PS.

3.9 Event Selection Function (S_E)

S_E selects an event and updates OS and μ in case it is an objective function and user preference respectively and in case it is a triggering event it passes it on to the interpreter which would unify it with the set of triggering events in the heads of plans.

3.10 Option Selection Function (S_O)

S_O selects a plan from P based on the current plan context, β, OS, PS and n. In order to understand the mechanism for selecting the best plan let us consider an example.

Let us consider we have two applicable plans - P1 and P2. In order to determine which plan to choose the agent generates the goal-plan tree for all possible paths. The parameter n creates the pseudo leafs and therefore we get distinct paths from root to these pseudo leafs. Figure 1 shows all the possible paths from root to pseudo leafs for the set of plans P1 to P10. The value of n is 2 which means the goal-plan tree is expanded up to 2 levels.

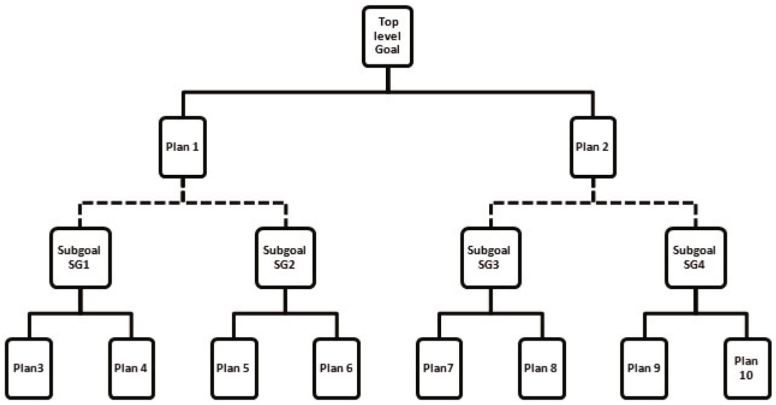
Multiple Effect Scenarios

Let us now consider any given path (say Path 1) in our earlier example. We follow [12] to discuss the issue of *multiple effect scenarios* in this context. The cummulative effect of this path is not merely a conjunction of the effects of Plan1, Plan3 and Plan5. The effects of each plan are accumulated into cumulative effect annotations in a context-sensitive manner, such that the cumulative effect annotations associated with any plan would describe the effects achieved by the execution of plans were it to execute up to that point. Since we are trying to find out what would be the final effect if we took Path 1 we use multiple effect scenarios as we cannot find the result deterministically. An *effect scenario* at a given point in a path is one consistent set of cumulative effects of a process if it were to execute up to that point. This is because we may arrive at a given point through multiple paths which cannot be predicted at design time and also activities in a path might *undo* the effects of activities earlier in the path.

Let $< P_i, P_j >$ be an ordered pair of plans connected via a sequence flow such that P_i precedes P_j , let ε_i be an effect scenario associated with ε_i and ε_j be the immediate effect annotation associated with P_j . Let $\varepsilon_i = c_{i1}, c_{i2}, \cdots, c_{im}$ and $e_j = c_{j1}, c_{j2}, \cdots, c_{jn}$. If $\varepsilon_i \bigcup \varepsilon_j$ is consistent, then the resulting cumulative effect, denoted by $acc(\varepsilon_i, \varepsilon_j)$, is $\varepsilon_i \bigcup \varepsilon_j$. Else, we define $\varepsilon'_i \subseteq \varepsilon_i$ such that $\varepsilon'_i \bigcup \varepsilon_i$ is consistent and there exists no ε''_i such that $\varepsilon'_i \subset \varepsilon''_i \subseteq \varepsilon_i$ and $\varepsilon''_i \bigcup \varepsilon_j$ is consistent. We define $acc(\varepsilon_i, \varepsilon_j) = \varepsilon'_i \bigcup \varepsilon_j$. We note that $acc(\varepsilon_i, \varepsilon_j)$ is non-unique as there are multiple alternative sets that satisfy the requirements for ε_i. Thus the cumulative effect of the two plans consists of the effects of the second plan plus as many of the effects of the first plan as can be consistently included. We remove those clauses in the effect annotation of the first plan that contradict the effects of the second plan. The remaining clauses are undone, i.e., these effects are overridden by the second plan.

Now each of these effects may have semiring value associated with them and hence we have get the semiring value for the cumulative effects. If PS contains $(< \varepsilon_p, v_p >, < \varepsilon_q, v_q >)$ then the semiring value of $acc(\varepsilon_p, \varepsilon_q)$ would be $v_p \bigotimes v_q$ where $\bigotimes$ is the semiring combination operator. Since we have multiple effect scenarios, the user would have the option to select a particular scenario which can be pessimistic or optimistic. Thus from each of the paths, a particular effect scenario is chosen and we get a semiring value for the cumulative effect at each pseudo leaf.

Plan1: $+!\text{t}[\varepsilon_1] : BContext_1 \cup CContext_1 \leftarrow SG_1; SG_2.$
Plan2: $+!\text{t}[\varepsilon_2] : BContext_2 \cup CContext_2 \leftarrow SG_3; SG_4.$
Plan3: $+!SG_1[\varepsilon_3] : BContext_3 \cup CContext_3 \leftarrow a_1.$
Plan4: $+!SG_1[\varepsilon_4] : BContext_4 \cup CContext_4 \leftarrow a_2.$
Plan5: $+!SG_2[\varepsilon_5] : BContext_5 \cup CContext_5 \leftarrow a_3.$
Plan6: $+!SG_2[\varepsilon_6] : BContext_6 \cup CContext_6 \leftarrow a_4.$
Plan7: $+!SG_3[\varepsilon_7] : BContext_7 \cup CContext_7 \leftarrow a_5.$
Plan8: $+!SG_3[\varepsilon_8] : BContext_8 \cup CContext_8 \leftarrow a_6.$
Plan9: $+!SG_4[\varepsilon_9] : BContext_9 \cup CContext_9 \leftarrow a_7.$
Plan10: $+!SG_4[\varepsilon_1 0] : BContext_{10} \cup CContext_{10} \leftarrow a_8.$



Broken line **(- - -)** refers to AND nodes and Solid line (——) refers to OR nodes.
$BContext_i$ is the conjunction of non-constraint predicates in the context of Plan$i$;
$CContext_i$ is the conjunction of constraint predicates in the context of pPan$i$;
SG_i is subgoal for Plan$i$;
a_i is an atomic action for Plan$i$;
ε_i is the effect of Plan$i$;

Path id	Possible Path
1	Plan1-Plan3-Plan5
2	Plan1-Plan4-Plan5
3	Plan1-Plan3-Plan6
4	Plan1-Plan4-Plan6
5	Plan2-Plan7-Plan9
6	Plan2-Plan8-Plan9
7	Plan2-Plan7-Plan10
8	Plan2-Plan8-Plan10

Fig. 1. Agent Plans and corresponding goal-plan tree

The next item to consider is the OS where we have the set of consistent objective functions that the agent wants to pursue. At each pseudo leaf of the goal plan tree we have

- β and context of all plans in the path.
- Semiring value for the effect scenarios.
- Objective function O.

There are several choices now. First, the CLP uses constraint satisfaction and optimization problem (CSOP) techniques to find the value of the objective function O for each pseudo leaf. CSOPs are mathematical problems where one must find states or objects that satisfy a number of constraints or criteria and also satisfy an objective function. Second, the semiring value v of cumulative effect could be considered for each of these leafs. The leaf to be considered would be the one which has both these values as the highest. It is a policy that the user can decide beforehand to give priority to either the objective or the semiring value as these values are nothing but natural means of representing softgoals that the agent would like to achieve. Next we choose the path which has the maximum value at the pseudo leaf based on the strategy chosen.

3.11 Intention Selection Function (S_I)

S_I function selects one of the agent's intentions, i.e., one of the independent stacks of partially instantiated plans within the set of intentions by applying techniques similar to that of S_O. Each intention stack is a choice for the agent. In order to determine right intention, the agent first considers the top element (i.e., a partially instantiated plan) of every intention stack. Each of these intentions for a goal plan tree and here also we solve the CSOP which consists of β, OS, PS and n. Like before, we have several choices and we can apply different strategy to choose a particular intention. However, one of the most notable difference with S_O is that we *instantiate the action parameters* (if any) with values obtained from solving the CSOP.

4 Differences with AgentSpeak(L)

Most of the syntax and semantics of BAOP are similar to that of AgentSpeak(L) and its interpreter Jason. However, the most notable additions are:

1. A constraint directed technique is incorporated into the computation strategy employed during the interpretation process.
2. Plan context consists of conjunction of predicates some of which could be constraint predicates (unlike Agentspeak(L)) which could be dealt with CLP machinery using specialized constraint solvers.
3. A look-ahead technique is now built into the system that helps the user to determine which particular plan or intention to select by setting the value of a look-ahead parameter.

4. An external event can be a triggering event as well as an addition or subtraction of an *objective function* or a *user preference*.
5. Two new data structures are added - an *objective store* and a *preference store* to store the set of global objectives and preferences respectively.
6. Plans are now annotated with effects. Jason provides annotation facility where meta information of various kinds could be specified. We use this facility to specify effects of plans.
7. Unlike AgentSpeak(L), applicable plans are those relevant plans for which
   - there exists a substitution which, when composed with the relevant unifier and applied to the context, is a logical consequence of β *and*
   - the constraint predicates in the context of the plans are unified with β and dealt with the CLP machinery using specialized constraint solvers to determine if these are consistent.
8. The set of basic *actions* that the agent has to perform as part of an intention execution process may also contain parameters, the values of which may be set by the value of the constraint variables obtained from solving of a CSOP relevant to a given applicable plan. These values are instantiated during intention execution.

Items 1 to 3 and parts of items 4 and 5 above have been developed in CASO. In BAOP we extend CASO to incorporate the rest of the items.

5 BAOP Interpreter

The BAOP interpreter is depicted in Figure 2 which greatly facilitates the understanding of the interpreter. It is very similar to the AgentSpeak(L) interpreter with the differences being mainly in handling of the seletion functions. The interpreter manages a set of events, a preference store, an objective store and a set of intentions with three selection functions which are described below. In the figure, sets (of beliefs, events, plans, preference store, objective store and intentions) are represented as rectangles, diamonds represent selection (of one element from a set) and circles represent some of the processing involved in the interpretation of BAOP programs. A CLP solver is plugged into the system which is responsible for generating the applicable plans as well as for the option and intention selection functions.

6 Representing Beliefs and Plans

Let us take a simple example where I have to buy orange juice and milk for tomorrow. I need to buy at least 4 lt. of milk(M) and 2 lt. of orange juice(OJ). Let the amount of money(A) I have is \$50. The cost of milk is \$2/lt. and orange juice is \$4/lt. I cannot buy more than 10 lt. in total as there is no space in the fridge. How much of each should I buy and what should be the total cost(C) if I want to have the maximum amount of money left on me? The problem can

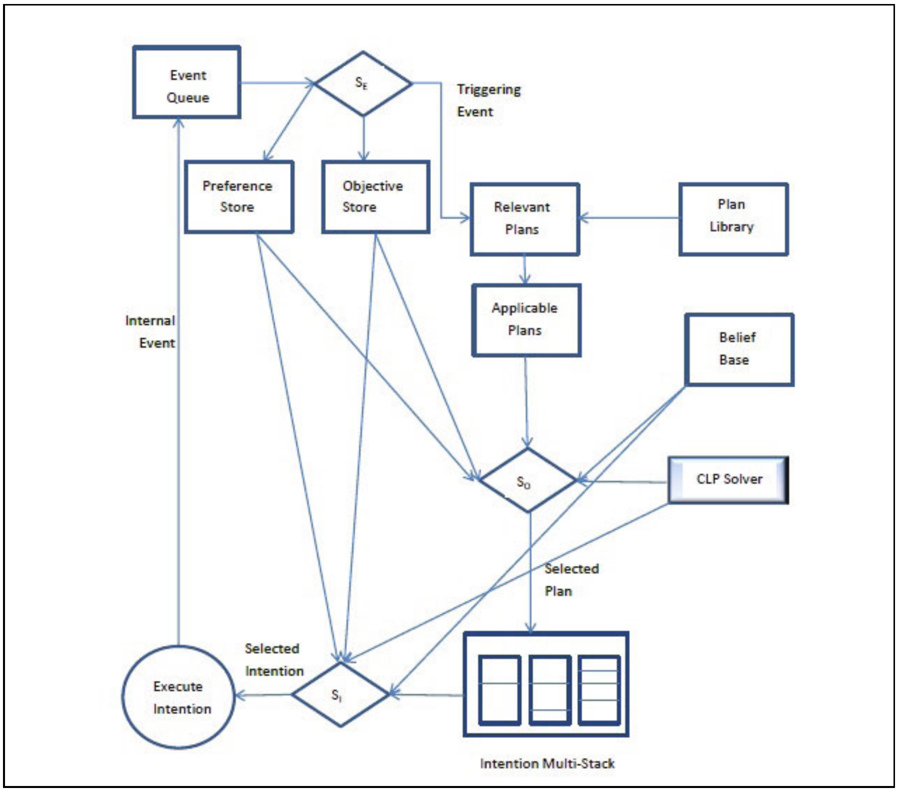


Fig. 2. BAOP execution cycle

easily be solved using constraint logic programming techniques. This problem can be formulated as follows:

```
A=50; y >= 2; x >= 4; x + y <= 10;
M>=4;  OJ>=2;  M + OJ <=10; C = 2*M-4*OJ;
maximize(A-C);
```

The above CSOP is fairly trivial and the result is M=4, OJ=2 and C=16. Hence the money left after buying these items is $34.

In the agent context, the above could be a belief base denoted by *moneyAvailable()*. Let us consider that I want to go either to a concert or go to a movie. Which one to choose may depend on a number of factors: the amount of money I have to buy ticket, the availability of the seats of my choice etc. As an example, for a concert I may want to choose front seats but for a movie I may want to choose back seats at the theater. In any case, if I want to go, I have to first book tickets over the phone. Also, let us assume that I have to buy milk and orange juice. All the above can be described using BAOP plans which would in turn have a series of subplans and basic actions. As an example, some of the plans (denoted by P1, P2 and P3) related to the concert or movie booking could be written as follows where A refers to the concert or movie and V refers to the venue:

```
***** Booking tickets related plans *****

P1::+concert (A,V) [BookedConcertTickets(V)] :
    likes(A) & moneyAvailable()>20 <- !book_tickets(A,V).

P2::+movie (A,V) [BookedMovieTickets(V)] :
    likes(A) & moneyAvailable()>30 <- !book_tickets(A,V)

P3::+!book_tickets(A, V)[called(V) & SeatsChosen(A,V)] :
    -busy(phone) <- call(V); !choose_seats(A,V).
...

***** Buying milk and orange juice related plans *****

P10::+!buy_MilkandJuice()[boughtMilkandJuice] :
    true <- get_milk(actionParam M); get_juice(actionParam OJ);
            pay_amount(actionParam C).
```

Note that *concert (A,V)* and *book_tickets(A, V)* are the triggering events; *likes(A)*, and *-busy(phone)* are contexts; *!book_tickets(A,V)* and *!choose_seats (A,V)* are subgoals; *call(V)* is a basic action and *BookedMovieTickets(V)*, *BookedConcertTickets*, *called(V)* and *shallChooseSeats(A,V)* are the various effects for plans related to booking tickets. *M*, *OJ* and *C* refer to the *action parameters* which we shall discuss shortly. The constraints are denoted by *moneyAvailable()*> 20 and *moneyAvailable()*> 30 which are part of the context.

The example above basically says that if I have more than \$20 on me and there is a movie playing at a venue and I like the movie then I will book the tickets over the phone and choose seats. Similarly for concert tickets I have to have \$30 on me and rest of the terms and conditions remain the same. It must be mentioned here that plans P1 and P2 are very similar - one refers to the process of booking concert tickets and the other one movie tickets. In normal circumstances, I would choose either of the two if all the conditions match - however I may have a preference of going to the concert today rather than to the movie. This preference is something that is not built into the plan but can be supplied by the user at any point in time to help make the decision. This can be done by giving quantitative values to the effects - say $< BookedMovieTickets(V), 0 >$ and $< BookedConcertTickets(V), 1 >$ to imply the fact that I prefer booking concert tickets to booking movie tickets.

For buying milk and orange juice we have plan P10. This plan has *true* in the context which means this will always be executed. Notice the parameters of the actions get_milk(), get_juice() and pay_amount(). The *actionParam M, OJ* and *C* simply state that I have to buy M lt. of milk, OJ lt. of orange juice and pay \$C.

7 BAOP: Operational Semantics

The operational semantics of BAOP is defined using Plotkin's Structural Operational Semantics [16]. We use a method similar to that shown in [15] which describes operational semantics of AgentSpeak(L). A BAOP *configuration* C is a tuple $C = \langle \beta, I, E, A, R, Ap, OS, PS, \iota, \epsilon, \rho, \alpha \rangle$ where

- β is a set of beliefs
- I is a set of intentions $\{i, i', \cdots\}$ and i is a stack of partially instantiated plans
- E is a set of events $\{(te, i), (oe), (sge), (te', i'), (oe'), (sge') \cdots\}$ where (1) *a triggering event pair* is denoted by (te, i) where te is the triggering event and the intention i has the plans associated with it; (2) *an objective event* is denoted by oe which adds or removes element from the objective store; (3) *a softgoal event* is denoted by sge which is a c-semiring structure with possible states and their corresponding values.
- A is a set of actions $\{(a_1, param_{1'}, \cdots), (a_2, param_{2'}, \cdots), \cdots\}$. Each action is a tuple $(a_i, param_{i'}, \cdots)$ where a_i is the basic action and $param_{i'}$, $param_{i''}$ etc. are the action parameters.
- R is a set of relevant plans.
- Ap is a set of applicable plans.
- OS is the objective store
- PS is the preference store
- Each configuration has 4 components denoted by ι, ϵ, ρ and α which keep record of a particular *intention*, *event*, an *U-preferred* plan (the selected plan with user preference) and a set of *action parameters* associated with actions of an U-preferred plan respectively that are being considered along the execution of a plan.

In order to present the semantic rules for BAOP we adopt the following notations:

- If C is a BAOP configuration, we write C_E to make reference to the component E of C. Similarly for other components of C.
- We write $C_\iota = \_$ (the underline symbol) to indicate that there is no intention being considered in the agent's execution. Similarly for C_ρ, C_ϵ and C_α.
- We write $i[p]$ to denote the intention that has plan p on its top.
- We write *beliefs* to denote the set of current beliefs together with the set of constraints.

The set of semantic rules related to Event, Plan and Intention selections are now given below.

Event Selection: S_E selects events from a set of Events E. The selected event is removed from E and is either assigned to the component ϵ of the configuration in case it is a triggering event or is added or removed to/from OS/PS if it is a objective/preference. The selected event is removed from E and is assigned to the ϵ component of the configuration. Below we give the three semantic rules governing this function.

$SelEv_1$ $\frac{S_E(C_E)=(te,i)}{C,beliefs \to C',beliefs}$ $C_\epsilon = \_, C_{AP} = C_R = \{\}$
where: $C'_E = C_E - (te, i)$ and $C'_\epsilon = (te, i)$

$SelEv_2$ $\frac{S_E(C_E)=oe}{C,beliefs \to C',beliefs}$ $C_\epsilon = \_, C_{AP} = C_R = \{\}$
where: $C'_E = C_E - oe,$ $C'_{OS} = C_{OS} + oe$ and $C'_\epsilon = oe$

$SelEv_3$ $\frac{S_E(C_E)=sge}{C,beliefs \to C',beliefs}$ $C_\epsilon = \_, C_{AP} = C_R = \{\}$
where: $C'_E = C_E - sge,$ $C'_{PS} = C_{PS} + oe$ and $C'_\epsilon = sge$

Option Selection: S_O selects an U-preferred plan (p) from the set of applicable plans Ap. The plan selected is then assigned to the ρ of the configuration and the set of applicable plans is discarded. We also assume that there is a selection function S_{OS} which selects a consistent objective store (ConsOS) where the maximal set of objectives that are consistent are kept and the rest are discarded. The action parameters associated with the U-preferred plan are initialized by solving the relevant CSOP in case there are hard constraints and objectives.

$SelOpr$ $\frac{S_O(C_{Ap})=p \quad S_{OS}(C_{OS})=ConsOS}{C,beliefs \to C',beliefs}, C_\alpha = \_, C_\epsilon \neq \_, C_{Ap} = \{\}$
where: $C'_\rho = p,$ $C'_{OS} = ConsOS,$ $C'_\alpha \neq \{\}$ and $C'_{Ap} = \{\}$

Creating Intentions: Rule TrigEv says that if the event ϵ is external triggering event indicated by T in the intention associated to ϵ, a new intention is created and its single plan is the plan p annotated in the ρ component. If the event is internal, rule IntEv says that the plan in ρ should be put on top of the intention associated with the event. Either way, both the event and the plan can be discarded from the ϵ and ι components respectively.

$TrigEv$ $\frac{}{C,beliefs \to C',beliefs}$ $C_\epsilon = (te, T), C_\rho = p$
where: $C'_I = C_I \bigcup \{[p]\},$ $C'_\epsilon = \_,$ $C'\rho = \_$
$IntEv$ $\frac{}{C,beliefs \to C',beliefs}$ $C_\epsilon = (te, T), C_\rho = p$
where: $C'_I = C_I \bigcup \{i[p]\},$ $C'_\epsilon = \_,$ $C'\rho = \_$

Intention Selection: S_I selects an intention for processing.

$SelInt$ $\frac{S_I(C_I)=i}{C,beliefs \to C',beliefs}$ $C_i = \_$
where: $C'_i = i$

8 Experiment and Results

We implemented BAOP by extending CASO and incorporating a mechanism by which user preferences can be added as soft constraints into the system. As mentioned earlier, BAOP has been developed using a modified version of Jason (Java based interpreter) and Eclipse (constraint solver). In order to evaluate our approach, we ran a series of experiments to test out the various scenarios described earlier pertaining to option selection using constraints, objectives and user preferences. The objective of our experiment was to find out if BAOP is capable of handling the different situations and choose the right plan and intention at every interpreter cycle.

We randomly generated several plans having a variety of AND-OR node structure. Each of these plans included, effects and the semiring values (user preferences) were set for each of the effects. We also defined partial order among the various effects The following table depicts the various experimental runs that we conducted on a Pentium Dual Core 2.4GHz CPU with depth of tree kept at 3. If the environment had inconsistencies between user preferences and objectives, the objectives were given higher precedence. Thus at each node both objective and user preferences were evaluated.

Plans	Possible Paths	CSOPs solved	time taken(millisec.)
10	8	8	100
20	15	0	78
30	22	14	134
40	34	20	146
50	40	25	178

The results indicate that the larger the number of CSOPs, the longer it takes to evaluate and get the right plan. If there are user preferences only and no CSOP, the time taken to find the selected plan was the least.

9 Related Work

Many of the other BDI languages have been extended to incorporate constraints and preferences to guide the agent to select the best possible plan.

In [8] a system that allows the user to express all these kinds of constraints and preferences is described by extending GOLOG [14]. The authors address the problem of combining non-Markovian qualitative preferences, expressed in first-order temporal logic, with quantitative decision-theoretic reward functions and hard symbolic constraints in agent programming. In another approach [21], prioritized goals have been integrated with the IndiGolog agent programming language.

In our approach the semiring specifications are richer than their specification of prioritized goals. Also, we combine both qualitative and quantitative constraints.

In [11] an approach has been made to extend GOAL (a language based on propositional logic) with temporal logic for the representation of goals and preferences for additional expressiveness. In this approach hard and soft constraints have been integrated into it as well as as achievement goals, maintenance goals, and temporally extended preferences. Here if an agent has the ability to lookahead a number of steps, it can also use its goals to avoid selecting those actions that prevent the realization of (some of) the agents goals. Effects of basic actions are taken into account which change the mental state of agents. The work is based on the fact that planners could be guided to select preferred plans.

One of the major differences with regards to our work is that the preferences and objectives are dynamic - they work like belief updates when the user his changes mind. Also, unlike other approaches, the hard constraints that we talk about here are objective functions in the current environment and not the agent goals.

In another approach [4], the authors describe a model where preferences and constraints over goals can be specified. They also mention an algorithm PREFPLAN for solving the resulting constrained optimization problem. PDDL3 [9] is an action-centred language in the planning domain where strong and soft constraints on plan trajectories (possible actions in the plan and intermediate states reached by the plan), as well as strong and soft problem goals has been incorporated.

One of the differences between agent programming and planning is that in planning one compares complete plans against the available constraints, while in agent programming the constraints are into account continuously during execution. In this context, the above frameworks deal mainly with plan uncertainty whereas our approach has been to incorporate similar notions into BDI agent paradigm.

There are a number of frameworks which mix planning and BDI-style execution. Of particular interest is CANPLAN [19] which have built-in capacity for *lookahead*.

In contrast, our *lookahead* mechanism is domain independent and is built into the BDI architecture. Moreover, when to do a lookahead is dependent on the user and in a highly reactive system, full lookahead may not give the best possible outcome.

10 Conclusion

While there has been some work in guiding an agent to select the best plan based on preferences, our approach is different from the ones mentioned earlier. Unlike most other work, we assume that our agent environment is highly reactive in nature and therefore constraints, preferences and objectives could change at every point in time. We give the user the ability to change objectives and preferences thereby making this a highly flexible system. In this work, we developed our agent programming language BAOP to incorporate preference relations. The preference relation is modelled using c-semirings. We applied the techniques to help the agent make decisions on selecting a particular plan at any given point of time. We also defined the formal semantics of BAOP as well as described algorithm with lookahead techniques for selecting the best plan. Our experiments show that the reactive property of BAOP agents are still maintained even when decisions are taken to find the path to take in order to achieve a particular goal. We have not paid much attention to the constraint solving techniques to find the best algorithm so that we can develop good elicitation strategies that reduce the time to find out the preferred plan quickly.

The preferences mentioned in this paper depict the relationship preference among the set of activities that are part of each plan as well as the objective functions. This notion could be further extended in a multi-agent environment where each agent would have a set of preferences and there could be a global preference set that would try to achieve an overall system optimization across all agents. This is also particulary useful in agent negotiation where each agent tries to negotiate in order to achieve its own goal.

References

1. Apt, K.R., Wallace, M.: Constraint Logic Programming using Eclipse. Cambridge University Press, New York (2007)
2. Bistarelli, S., Montanari, U., Rossi, F.: Semiring-based constraint satisfaction and optimization. Journal of ACM 44, 201–236 (1997)
3. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak using Jason. Wiley Series in Agent Technology. John Wiley & Sons, Chichester (2007)
4. Brafman, R.I., Chernyavsky, Y.: Planning with goal preferences and constraints. In: Biundo, S., Myers, K.L., Rajan, K. (eds.) Proceedings of the 15th International Conference on Automated Planning and Scheduling, pp. 182–191. AAAI, Menlo Park (2005)
5. Busetta, P., Ronnquist, R., Hodgson, A., Lucas, A.: JACK intelligent agents - components for intelligent agents in java. In: AgentLink News Letter, Agent Oriented Software Pty. Ltd. (January 1999)
6. Dasgupta, A., Ghose, A.K.: CASO: a framework for dealing with objectives in a constraint-based extension to AgentSpeak(L). In: Proceedings of the 29th Australasian Computer Science Conference, ACSC 2006, Darlinghurst, Australia, vol. 48, pp. 121–126. Australian Computer Society, Inc. (2006)
7. Dasgupta, A., Ghose, A.K.: Implementing reactive BDI agents with user-given constraints and objectives. Int. J. Agent-Oriented Software Engineering 4, 141–154 (2010)
8. Fritz, C., Mcilraith, S.A.: Decision-theoretic golog with qualitative preferences. In: Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR), Lake District, UK, June 25, pp. 153–163 (2006)
9. Gerevini, A., Long, D.: Plan constraints and preferences in PDDL3. Technical report, Dipartimento di Elettronica per l'Automazione, Universit di Brescia (2005)
10. Hindriks, K.V., de Boer, F.S., der Hoek, W.V., Meyer, J.-J.C.: Agent programming in 3APL. In: Autonomous Agents and Multi-Agent Systems, Hingham, MA, USA, November 1999, vol. 2, pp. 357–401. Kluwer Academic Publishers, Dordrecht (1999)

11. Hindriks, K.V., Birna Riemsdijk, M.: Using temporal logic to integrate goals and qualitative preferences into agent programming. In: Baldoni, M., Son, T.C., Birna Riemsdijk, M., Winikoff, M. (eds.) DALT VI 2008. LNCS (LNAI), vol. 5397, pp. 215–232. Springer, Heidelberg (2009)
12. Hinge, K., Ghose, A.K., Koliadis, G.: Process seer: A tool for semantic effect annotation of business process models. In: Proceedings of 2009 IEEE International Enterprise Distributed Object Computing Conference (EDOC 2009), pp. 54–63 (2009)
13. Jaffar, J., Maher, M.: Constraint logic programming: A survey. Journal of Logic Programming, Special 10th Anniversary Issue (19/20) (May/July 1994)
14. Levesque, H.J., Reiter, R., Lespérance, Y., Lin, F., Scherl, R.B.: Golog: A logic programming language for dynamic domains. Journal of Logic Programmming 31(1-3), 59–83 (1997)
15. Moreira, I.F., Bordini, R.H.: An operational semantics for a BDI agent-oriented programming language. In: Proceedings of the Workshop on Logics for Agent-Based Systems (LABS 2002), Toulouse, France, pp. 45–59 (April 2002)
16. Plotkin, G.D.: A structural approach to operational semantics. Technical Report DAIMI FN-19, University of Aarhus (1981)
17. Rao, A.S.: Agentspeak(L): BDI agents speak out in a logical computable language. In: Perram, J., Van de Velde, W. (eds.) MAAMAW 1996. LNCS, vol. 1038, pp. 42–55. Springer, Heidelberg (1996)
18. Rao, A.S., Georgeff, M.P.: BDI agents: From theory to practice. In: Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS 1995), San Fransisco, USA, pp. 312–319 (1995)
19. Sardina, S., de Silva, L., Padgham, L.: Hierarchical planning in BDI agent programming languages: A formal approach. In: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), pp. 1001–1008. ACM Press, New York (2006)
20. Sardina, S., Padgham, L.: Goals in the context of BDI plan failure and planning. In: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007), pp. 1–8. ACM, New York (2007)
21. Sardina, S., Shapiro, S.: Rational action in agent programs with prioritized goals. In: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2003), pp. 417–424. University Press, New Haven (2003)
22. Shapiro, S., Lespérance, Y.: Modeling Multiagent Systems with CASL - A Feature Interaction Resolution Application. In: Castelfranchi, C., Lespérance, Y. (eds.) ATAL 2000. LNCS (LNAI), vol. 1986, pp. 244–259. Springer, Heidelberg (2001)
23. van Riemsdijk, B., van der Hoek, W., Meyer, J.-J.C.: Agent programming in dribble: from beliefs to goals using plans. In: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2003), pp. 393–400. ACM, New York (2003)
24. Winikoff, M., Padgham, L., Harland, J., Thangarajah, J.: Declarative & procedural goals in intelligent agent systems. In: Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR 2002), pp. 470–481 (2002)
25. Wooldridge, M., Jennings, N.R.: Intelligent agents: Theory and practice. Knowledge Engineering Review 10(2), 115–152 (1995)