2006

CASO: A framework for dealing with objectives in a constraint-based extension to AgentSpeak(L)

Aniruddha Dasgupta
University of Wollongong

Aditya K. Ghose
University of Wollongong, aditya@uow.edu.au

CASO: A framework for dealing with objectives in a constraint-based extension to AgentSpeak(L)

Abstract

Incorporating constraints into a reactive BDI agent programming language can lead to better expressive capabilities as well as more efficient computation (in some instances). More interestingly, the use of constraint-based representations can make it possible to deal with explicit agent objectives (as distinct from agent goals) that express the things that an agent may seek to optimize at any given point in time. In this paper we extend the preliminary work of Ooi et.al in augmenting the popular Belief-Desire-Intention (BDI) language AgentSpeak(L) with constraint-handling capabilities. We present a slightly modified version of their proposal, in the form of the language CAS (Constraint AgentSpeak). We then extend CAS to form the language CASO (Constraint AgentSpeak with Objectives) to incorporate explicit objectives (represented as objective functions) and present techniques for performing option selection (selecting the best plan to use to deal with the current event) as well as intention selection. In both cases, we present parametric look-ahead techniques, i.e., techniques where the extent of look-ahead style deliberation can be adjusted.

Disciplines

Physical Sciences and Mathematics

Publication Details

Dasgupta, A. & Ghose, A. K. (2006). CASO: A framework for dealing with objectives in a constraint-based extension to AgentSpeak(L). In V. Estivill-Castro & G. Dobbie (Eds.), Australasian Computer Science Conference (pp. 121-126). Sydney, NSW: Australian Computer Society Inc.

CASO: A Framework for dealing with objectives in a constraint-based extension to AgentSpeak(L)

Aniruddha Dasgupta **Aditya K. Ghose**

Decision Systems Lab
School of IT and Computer Science
University of Wollongong,
Wollongong, NSW 2522,
Email: ad844@uow.edu.au, aditya@uow.edu.au

Abstract

Incorporating constraints into a reactive BDI agent programming language can lead to better expressive capabilities as well as more efficient computation (in some instances). More interestingly, the use of constraint-based representations can make it possible to deal with explicit agent objectives (as distinct from agent goals) that express the things that an agent may seek to optimize at any given point in time. In this paper we extend the preliminary work of Ooi et.al in augmenting the popular Belief-Desire-Intention (BDI) language AgentSpeak(L) with constraint-handling capabilities. We present a slightly modified version of their proposal, in the form of the language CAS (Constraint AgentSpeak). We then extend CAS to form the language CASO (Constraint AgentSpeak with Objectives) to incorporate explicit objectives (represented as objective functions) and present techniques for performing option selection (selecting the best plan to use to deal with the current event) as well as intention selection. In both cases, we present parametric look-ahead techniques, i.e., techniques where the extent of look-ahead style deliberation can be adjusted.

1 Introduction

The concept of using constraints has been introduced by Ooi *et al.* (1999) where it has been shown that the integration of constraints in a high-level agent specification language yields significant advantages in terms of both expressivity and efficiency. The BDI framework employed in the multi agent broker system is implemented with an improvised computation strategy - a synergy of unification and constraint solving. The improvisation applies constraint directed solving on the context section of a BDI agents plan specification in order to determine an application plan to fire. The constraint system introduced into the BDI framework maintains a constraint store that collects a set of constraints that augment the beliefs of an agent.
In this paper we extend the work one by Ooi *et al.* (1999) by incorporating explicit objectives beside the constraints. We also describe some efficient plan and intention selection methods which would result in better expressibility and more efficient computation which has not been addressed in either Agentspeak(L) introduced by Rao (1996) or by Ooi *et al.* (1999). This type of selection mechanisms are particulary useful in many real world applications which require the use of intelligent agents to perform some critical tasks. This paper extends the preliminary work presented by Dasgupta *et al.* (2005).
The remainder of this article is organized as follows. Section 2 gives an example which is used throughout the rest of the paper. Section 3 introduces the language CASO and section 4 discusses its operational semantics and describes the algorithms for efficient plan and intention selection. Finally, concluding remarks and comparisons are presented in the last section.

2 Motivation

In this section we give an example of detailed reasoning behind the adoption of CASO. We begin by outlining a specific scenario of using CASO by a truck in order to deliver goods one location to another. The roads that the truck would take consists of several roads with choices available at various important points to follow one of the many paths. For simplicity, let us assume that the truck can either take the city road or the highway and both runs in parallel and the truck can at exit from the highway into a city road or enter the highway from city road from the important points.

Let us assume that there following tasks that need to be achieved.
G1. Deliver a parcel X to location B from the current location A.
G2. Fill up the tank whenever there is less than a quarter of petrol in the tank.

The following objectives may also be supplied to the truck driver.
O1. Choose the shortest path for delivery of the parcel.
O2. Minimize the amount of petrol required.

A constraint the truck driver may be supplied with might be the following.
C1. Parcel must be delivered by 5p.m.

Let us also assume the following ground beliefs.
B1. Petrol consumption rate in highways is 10 k.m./litre.
B2. Petrol consumption rate in city roads is 8 k.m./litre.

The two goals above are fairly independent of each other. Within an agent context the above tasks may be represented as a set of goals that need to be fulfilled. In order to fulfil each goal, the truck driver needs to execute a sequence of actions (i.e. to execute a plan). There might be a number of plans for

achieving the same task. As an example, for achieving the first goal there might be two possible plans:
Plan P1: *1. From location A take H1. 2. Deliver the parcel X at B.*
Plan P2: *1. From location A take city road R1. 2. Deliver the parcel X at B.*
Note that each of the plans above may have subplans which would describe the exact route to be followed. Both the above plans achieve the same result of delivering the parcel. However, the difference that exist are the time and petrol needed. In case of plan P1, the time taken is less as there is less traffic and for plan P2, the amount of fuel required is less whereas time taken is more.

3 Agent Programming with CASO

Informally, an agent program in CASO consists of a set of beliefs B, a set of constraints C, an objective function O, a set of events E, a set of intention I, a plan library P, a constraint store CS, an objective store OS and three selection functions S_E, S_P, S_I to select an event , a plan and an intention respectively to process and n_p and n_i are the two parameters which denote the number of steps to look-ahead for plan and intentions selection respectively.

Definition 1:
An agent program is a tuple {*B, P,E, I,C,O,*S_O, S_E, S_I, n_p, n_i*,CS,OS*} *where*
B is a set of Beliefs.
P is agent plan repository, a library of agent plans.
E is set of events (including external and internal).
I is a set of intentions.
C is a set of constraints.
O is an objective function.
S_E *is a selection function which selects an event to process from set E of events.*
S_O *is a selection function which selects an applicable plan to a trigger t from set P of plans.*
S_I *is a selection function which selects an intention to execute from set I of intentions.*
CS is a constraint store which stores constraints which come as events.
OS is an objective store which stores the objective function which comes as an event.
n_p *is an integer which denotes the number of steps required to look-ahead for plan selection.*
n_i *is an integer which denotes the number of steps required to look-ahead for intention selection.*

In CASO, a constraint directed improvisation is incorporated into the computation strategy employed during the interpretation process. Constraint logic programming (CLP) combines the flexibility of logic with the power of search to provide high-level constructs for solving computationally hard problems such as resource allocation.
Formally, a language CLP(X) is defined by *a constraint domain X, a solver for the constraint domain X and a simplifier for the constraint domain X.*

Definition 2:
A CASO plan p is of the form $t : b_1 \wedge b_2 \wedge \cdots \wedge b_n \wedge c_1 \wedge c_2 \wedge \cdots \wedge c_m \leftarrow sg_1, sg_2, \cdots, sg_k$ *where t is the trigger; each* b_i *refers to a belief; each* c_i *is an atomic constraint; each* s_g *is either an atomic action or a subgoal.*
For brevity we will use *BContext(p)* to denote the belief context of plan.
Thus $BContext(p) \equiv b_1 \wedge b_2 \cdots \wedge b_n$
Similarly, we will use CContext(p) to denote the constraint context of plan p.
Thus $CContext(p) \equiv c_1 \wedge c_2 \cdots \wedge c_m$

In our trucking example the beliefs and plans could be given as follows where TF refers to 'Tank Full', FC to 'Full Capacity of tank' and CL to 'Current Level' :

Beliefs
TF = false.
FC = 60.

Plans
+!fill-tank(CL):
$TF = false \& FC = 60 \& CL < 0.25 \times FC \leftarrow$ *(stop-to-fill(gas-station)); delay(5).*

The above plan simply states that in order to achieve the goal of filling the tank, the tank has to be quarter full and the actions to be taken would to stop at a gas station and fill up the tank and this would have a delay of Transition of agent program to process events depends on the event triggers. An event trigger, t, can be addition(+) or removal(-) of an achievement goal($\pm g_i$) or a belief ($\pm b_i$).

4 Operational Semantics of CASO

The CASO interpreter manages a set of events, a constraint store, a objective store and a set of intentions with three selection functions. Intentions are particular courses of actions to which an agent has committed in order to handle certain events. Each intention is a stack of partially instantiated plans. Events, which may start off the execution of plans that have relevant triggering events, can be external when originating from perception of the agents environment (i.e., addition and deletion of beliefs based on perception are external events) ; or internal, when generated form the agents own execution of a plan (i.e., as subgoal in a plan generates an event of the type addition of an achievement goal).
In the latter case, the event is accompanied with the intention which generated it (as the plan chosen for that event will be pushed on top of that intention). External events create new intentions, representing separated focuses of attention for the agents acting on the environment.
The constraint store is initialized by the relevant constraints whenever a trigger contains a constraint in its context. At every cycle of the interpreter, the constraint store is enhanced with new constraints when applicable selected plan is executed. These incremental constraints collecting process eventually leads to a final consistent constraints set. Constraint solving is applied to the context of each plan to determine applicable plans as well as to generate solutions for subsequent actions. Similarly, the objective store contains the set of objective functions that need to be maximized (or minimized) which are part of the event context and is similarly updated at each cycle.
In the following sections we explain the basics of how CASO interpreter works. At every interpretation cycle of an agent program, CASO updates a list of events, which may be generated from perception of the environment, or from the execution of intentions (when subgoals are specified in the body of plans). It is assumed that beliefs are updated from perception and whenever there are changes in the agents beliefs, this implies the insertion of an event in the set of events.

4.1 Plan selection

After S_E has selected an event, CASO has to unify that event with triggering events in the heads of plans. This generates a set of all relevant plans. The constraints (if any) that are included in the constraint part of the context are put in the constraint store. The context part of the plans is unified against the agents beliefs. Constraint solving is now performed on these relevant plans to determine whether the constraint(s) in the context of the plan is (are) consistent with the constraints already collected in the constraint store . This results in a set of applicable plans(plans that can actually be used at that moment for handling the chosen event).
The objective store maintains a set of objective function which may be present in the event context. At each interpreter cycle, the objective store is also updated with an objective function for maximizing (or minimizing).

Definition 3:
Given plans p1 and p2 in the plan library, and given a current constraint store C and a current objective store O, $p_1 \leq_{opt} p_2$ *if and only if:* $OptSol(C \cup CContext(p_1), OS) \geq OptSol(C \cup CContext(p_2), O)$ *where Opt-Sol(Constraints,Objective) denotes the value of the objective function when applied to the optimal solution to the problem denoted by the pair (Constraints, Objective).*
We assume of course that $C \cup CContext(p_1)$ and $C \cup CContext(p_2)$ are solvable.

Optimization techniques are then applied by the optimizer to each of the applicable plan to determine an optimal solution. In effect we are solving a 'Constraint Satisfaction Optimisation Problem' (CSOP) which consists of a standard 'Constraint Satisfaction Problem' (CSP) and an optimisation function that maps every solution (complete labelling of variables) to a numerical value. S_O now chooses this optimal solution from that set, which becomes the intended means for handling that event, and either pushes that plan on the top of an existing intention (if the event was an internal one), or creates a new intention in the set of intentions (if the event was external, i.e., generated from perception of the environment). Thus plan selection is defined as follows:

Definition 4:
Given a trigger t and a set of applicable plans AppPlans(t) for t, a plan $p \in AppPlans(t)$ *is referred to as an O-preferred plan if and only if:* $p \leq_{opt} p_i$ *for all* $p_i \in AppPlans(t)$.

The agent program is also responsible for making sure that the objective store is consistent at any point of time. During each cycle of the interpreter, new objectives are added into the objective store and hence a consistency checker is used to maintain consistency. Formally a consistent objective store is defined as below.

Definition 5:
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*
1. x is consistent and
2. there exists no x' such that $x \subset x' \subseteq X$ *and x' is consistent.*
It is to be noted here that the triggering event can be the removal of an objective function also. The new objective store is now given by $\gamma(MaxCons(OS \cup \overline{O}) \cap OS$ where γ is the choice function, OS is the objective store and $\overline{O}$ is the negation of the objective O.

Selection of O-preferred plan can be further enhanced by using np the lookahead parameter form plan selection. In case np=0, no look-ahead is performed and maximizing the objective function on the set of applicable plans would result in an O-preferred plan as described earlier. However, if $n_p > 0$ then a look-ahead algorithm (similar to the one used for choosing the next move in a two-player game) is performed to select the O-preferred plan.
We assume that the agent is trying maximize its objective function and the environment may change in the worst possible way which would minimize the objective function. The goal of the agent would be to select a plan which would maximize the minimum value of the objective function resulting from the selection of plans which may occur due to the set of new possible events that may come from the environment.
We follow the definition of goal-plan tree given by Thangarajah (2004) to decompose the set of plans into a tree structure. In CASO, goals are achieved by executing plans and each goal has at least one plan, if not many, that can be used to satisfy the goal. Each plan can include sub-goals, but need not have any. The leaf nodes of the tree are plan-nodes with no children (i.e., no sub-goals).

Definition 6:
The relationship between a top level goal, its plans and subgoals defines a tree structure for each top-level goal, which is termed the goal-plan tree for that goal.

Each goal-plan tree consists of - a number of 'AND' nodes which are subgoals that must be executed sequentially for the goal to succeed; and a number of 'OR' nodes which are subgoals any one of which must be executed for the goal to succeed.

In our trucking example,there are two important criteria, which the user may want to satisfy:
1. the vehicle should go from the starting point to the destination point as fast as possible
2. the vehicle should go from the starting point to the chosen destination by maximum fuel saving.
The cost function for one length unit of a road R_i may look as: $C_u(R_i) = K \times F_u(R_i) + 1$ where $C_u(R_i)$ is the cost of one length unit (for example one meter) of the road $R_i, F_u(R_i)$ is fuel consumption for one length unit of the road R_i, and K denotes the degree of compromise (it must be a number equal or greater then zero). If K = 0 then the fuel consumption will be ignored and only the number of length units will be important the algorithm will find the shortest way to the destination. If the K parameter is a high number, the fuel saving will be very important for the optimization algorithm. The (global) cost of N used roads will then be the sum of N road costs: $TC = \sum(L(R_i) \times C_u(R_i))$. TC is the total cost of plan for the optimization algorithm and $L(R_i)$ is the used length of a road R_i.

Given a set of applicable plans, the truck agent would always try to achieve this objective at every decision step. However, there could be unforeseen road blocks and other situations which may result in the truck from changing its route at any of these decision points. This may result in the truck in spending more fuel than that what it would have

used. Thus the strategy for the agent is to compute in advance the worst case scenario that may occur due to the change in the highly dynamic environment. Let us consider that the example of two applicable plans p1 and p1 each having one subgoal.

Plan p_1
$+!location(truck, D1, k) : location(truck, R1) \& k \geq 0 \leftarrow !follow(A, F1, L1, k)$

Plan p_2
$+!location(truck, D1, k) : location(truck, R1) \& k \geq 0 \leftarrow !follow(B, F2, L2, k)$

p_1 suggests that if truck is at location R1 and it needs to go to destination D1 then it can follow route A. p_2 suggests an alternate route of going to D1 from R1 given by B. F1 and F2 are the fuel consumption per kilometer of distance and L1 and L2 are the lengths of the two roads and k is the fuel compromise factor as described earlier. Let us assume that plan p_1 and p_2 have the following possible subplans.

Plan $p_{1.1}$
$+!follow(A, F, L, k) : F = 3 \& L = 3 \& (timeleft < 1) \& k \geq 0 \& k \leq 2 \leftarrow +!drive(A)$

Plan $p_{1.2}$
$+!follow(A, F, L, k) : F = 1.5 \& L = 3 \& (timeleft > 1) \& k \geq 2 \leftarrow +!drive(A)$

Plan $p_{2.1}$
$+!follow(B, F, L, k) : F = 3 \& L = 2 \& (timeleft < 1) \& k \geq 0 \leftarrow +!drive(B)$

Plan $p_{2.2}$
$+!follow(B, F, L, k) : F = 0.5 \& L = 2 \& (timeleft < 1) \& k \geq 2 \leftarrow +!drive(B)$

Plan $p_{1.1}$ suggest that if current time left to reach destination is less than 1 hr. then, the value of k should lie between 0 and 2. Similarly, plan $p_{1.2}$ suggests k should be greater than 2 if more than 1 hr. of time is left. Plan $p_{2.1}$ and $p_{2.2}$ suggest similar plans for route B.

Since the objective is to maximize the value of TC shown earlier, let the constraint solving yields the value of k=0 from plan $p_{1.1}$ and k=2 for plan $p_{1.2}$. Similarly, for plan p_2, the values for k are 0 and 2 respectively. Figure 1 shows the tree decomposition for plan p depicting all possible choices. The numbers corresponding to the leaf nodes are the values of the optimization function TC which we are trying to maximize. Thus choosing plan $p_{1.1}$ and $p_{1.2}$ would give values 3 and 12 respectively; similarly for plans $p_{2.1}$ and $p_{2.2}$ the corresponding values would be 2 and 4. Using the *LookAheadPlanSelection* algorithm shown below, we obtain the value of 3 at the root node which suggest that the agent should follow plan p_1.

Following the above algorithm, the truck agent would choose the p_1.

4.1.1 Incremental Resolving of CSOPs

A single decision with such a strategy has $O(b^n)$ time complexity where b is the branching factor of the decision tree being explored and n is the number of steps to look ahead which is passed on as a parameter. The efficiency of plan selection can be greatly improved if we do not solve the CSOPs at every step from the beginning of n-step look-ahead at each decision point but instead apply some heuristic for incrementally resolving the CSOPs.

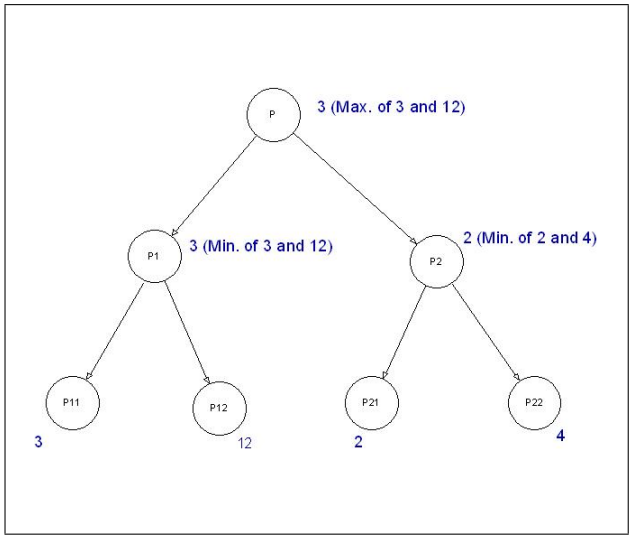

Figure 1: Plan Tree

Algorithm 1 LookAheadPlanSelection(int n, state S, ObjectiveStore OS, ConstraintStore CS)

1: Generate goal-plan tree up to n levels from current state S comprising of subgoals of AND and OR nodes with subplans.
2: Start from the root node.
3: Let constraint store at node $p = c_p$
4: Let o_p denote the value of objective function at node p.
5: For each node p in the goal plan tree set $c_p \leftarrow CS$
6: **if** node p has child nodes $p_1, p_2 \cdots, p_k$ in an AND structure **then**
7: Apply constraint solving at each p_i with the current constraint store cp_i and the set of constraints for p_i to obtain op_i.
8: Set $c_{pi+1} \leftarrow c_{pi}$ for all $i \geq 1$.
9: Initialize constraint store for all child nodes of each p_i with c_{pi}.
10: **end if**
11: **if** node p has child nodes $p_1, p_2 \cdots, p_k$ in an OR structure **then**
12: Compute the objective function and update the constraint store for each p_i.
13: Initialize constraint store for all child nodes of each p_i with c_{pi}.
14: **end if**
15: **while** $n \neq 1$ **do**
16: Propagate minimum value of objective function up to each parent node starting from the leaf node.
17: $n = n - 1$
18: **end while**
19: Propagate the maximum value of its children for state S.
20: At state S, the best plan is the child with the maximum value.

A heuristic similar to *Look Back* schemas like *backtracking* that are often used for consistency check in CSPs can be employed for resolving the CSOPs. Without any look-ahead, the CSOP for the given plan P is solved and the solution along with the set of constraints in the constraint store at that point is stored. In order to solve the CSOP for each step of look-ahead, the new set of constraints that are associated with each of the subplans at each decision point for plan P is added to the currently solved CSOP set -

if this new set of constraints violate the current value of the objective function, then backtracking is performed to the most recently instantiated variable that still has alternatives available and the new CSOP is solved with the new value of the instantiated variable.

4.2 Intention Selection and Execution

Once a plan is chosen the next stage is to execute a single intention in that cycle. The S_I function selects one of the agents intentions (i.e., one of the independent stacks of partially instantiated plans within the set of intentions). Look-ahead technique using decision tree is similarly employed here which could help in selecting an intention which would give the optimal solution. The parameter n_i denotes the number of steps for required to look ahead. In case of Intention selection, this merely becomes the number of items to be evaluated at the top of the intention stack. If there are more than one intention stacks present, then look-ahead procedure pops the top n_i elements of the stack from each intention and computes the optimal solution based on the constraint and the objective store.
Let us assume that in case of our truck agent, there are currently two intention stacks (I_1 and I_2) each corresponding to the two independent goals. The first one is to follow plan p_1 (described earlier) and the other one is to follow plan p_3 which describes the goal of picking up a parcel P2 from location C.

1. *Take route A to location D1 from* R1 *(plan* p_1*).*
2. *Pick up parcel P2 from location C (plan* p_3*).*

Both the above constitute a set of plans which are in the intention stack ready to be executed. The deliberation process of the truck agent is to decide which intention stack to pursue at a given point in time. For our example, let us assume that plan p_1 has subplan p_{11} in the intention stack (i.e. S_O has selected plan p_{11} to be executed). Let us also assume that the body of plan p_3 consists of the subplan p_{31} followed by subplan p_{32} followed by action a_1 (i.e. $p_{31}; p_{32}; a_1$).
The intention selection mechanism with one step look ahead would be to enumerate each of the above plans p_1 and p_3 with respect to the set of constraints associated with each plan and objective function (i.e. *Maximize TC*) to determine the best intention to execute. A two-step look-ahead mechanism would look at maximizing the value of TC by enumerating plans p_{11} and p_{31} with the set of constraints associated with

1. plan p_1 along with subplan p_{11} on intention stack I_1;
2. plan p_3 along with subplan p_{31} on intention stack I_2.

Thus depending on n_i, the *number of steps* given by the programmer, the prioritization of intention is determined by the value of the objective function up to n_i levels in the intention stack for each intention. The one which yields the maximum value of the objective function would the the intention selected by S_I. In this case the tree generated is called the intention tree as shown in Figure 2 below where $I_1, I_2, \cdots I_n$ are the set of intention stacks.
The algorithm for selecting new intention using look-ahead is given below.

On the top of the selected intention there is a plan, and the formula in the beginning of its body is taken for execution. This implies that either a basic action is performed by the agent on its environment, an internal event is generated (in case the selected formula

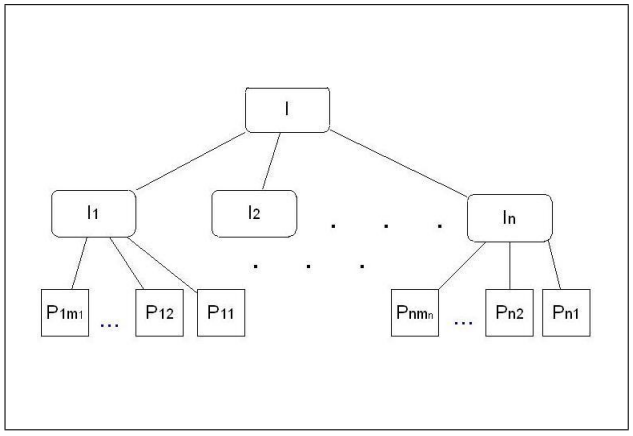

Figure 2: Intention Tree

Algorithm 2 LookAheadIntentionSelection(int n, ObjectiveStore OS, ConstraintStore CS))

1: Generate intention tree for all Intention Stacks.
2: Let constraint store at node $p = c_p$
3: Compute the value of objective function at the leaf nodes starting from left to right for each intention stack up to n nodes (i.e. n elements from top of Intention Stack) each by taking objectives and constraints from the objective store and the constraint store.
4: The best intention to execute is the intention stack which has the maximum value of the objective function at node n from the left.

is an achievement goal denoted by $!g_i$), or a test goal is performed (which means that the set of beliefs has to be checked). If the intention is to perform a basic action or a test goal denoted by $?g_i$, the set of intentions needs to be updated. In the case of a test goal, the belief base will be searched for a belief atom that unifies with the predicate in the test goal. If that search succeeds, further variable instantiation will occur in the partially instantiated plan which contained that test goal (and the test goal itself is removed from the intention from which it was taken). In the case where a basic action is selected, the necessary updating of the set of intentions is simply to remove that action from the intention (the interpreter informs to the architecture component responsible for the agent effectors what action is required). When all formulae in the body of a plan have been removed (i.e., have been executed), the whole plan is removed from the intention, and so is the achievement goal that generated it (if that was the case).
This ends a cycle of execution, and CASO starts all over again, checking the state of the environment after agents have acted upon it, generating the relevant events, and so forth.

5 Comparison and Conclusion

We now briefly summarize some of the work related to AgentSpeak(L) and BDI framework below.
Chalmers *et al.*(2001) constraint logic programming and data model approach is used within BDI agent framework. However, this work speaks of BDI agents in general and does not integrate with any BDI programming language. AgentSpeak(XL) programming language as described by Bordinin *et al.*(2002) integrates AgentSpeak (L) with the TAEMS scheduler in order to generate the intention selection function. It also describes a precise mechanism for allowing pro-

grammers to use events in order to handle plan failures which is not included in AgentSpeak(L). This work, however, adds priority to the tasks. Some related theoretical work on selecting new plans in the context of existing plans is presented by Horty *et al.*(2001). Another related work on detecting and resolving conflicts between plans in BDI agents is presented by Thangarajah *et al.*(2003). The degree of boldness of an agent, as defined by Schut *et al.*(2000), represents he maximum number of plan steps the agent executes before re-considering its intentions. However in this case it is assumed that the agent would backtrack if the environment changes after it has started executing the plans.
In this paper we have presented a general overview and informal discussion of the concept of incorporating constraints and objectives functions to AgentSpeak(L) as well as describe a means of how to design the option selection function for selecting a plan or an intention by using parametric look ahead mechanism. In future we would be extending CASO to incorporate inter-agent constraints in a multi-agent environment where agents may need to negotiate with each other.

References

Ooi, B. Hua & Ghose, A. K. (1999), Constraint-Based Agent Specification for a Multi-agent Stock Brokering System, *in* 'Multiple Approaches to Intelligent Systems:Proceedings of the 12th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems', Vol. 1611, Springer-Verlag Lecture Notes in Computer Science, pp. 409–419.

Jaffar, J. & Maher, M.J. (1994), Constraint logic programming: A survey., *in* 'Journal of Logic Programming' pp. 503–581

Kinny, D. & Georgeff, M. (1997), Modeling and design of multi-agent systems, *Intelligent Agents III, Lecture Notes in Artificial Intelligence Springer*, Berlin.

Morley., D. (1996), Semantics of BDI agents and their environment., *in* 'Tech. Rep. 74, Australian Artificial. Intelligent Institute, Melbourne'.

Rao, A.S. (1996), AgentSpeak(L): BDI agents speak out in a logical computable language, *in* 'Agents Breaking Away: Proceedings of the 7th European WS on Modelling Autonomous Agents in a Multi-Agent World', LNAI Vol 1038, pp. 42–55, Springer Verlag: Heidelberg, Germany.

Rao, A.S. & Georgeff, M. (1995), BDI Agents: from theory to practice., *in* 'Proceedings of First International Conference on Multi-Agent Systems', ICMAS-95, pp 312–319, San Francisco, CA.

Schut, M. & Wooldridge, M. (2000), Intention reconsideration in complex environments, *in* 'Proceedings of International Conference on Autonomous Agents', Barcelona, Spain.

Bordini, R.H., Bazzan, A.L.C., Jannone, R.O., Basso, D.M., Vicari, R.M. & Lesser, V.R. (2002), AgentSpeak(XL):Efficient intention selection in BDI agents via decisiontheoretic task scheduling, *in* Castelfranchi C. & Johnson, W.L. eds, 'Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems', ACM Press, pp 1294–1302, New York, USA.

Horty, J. & Pollack, M. (2001), Evaluating new options in the context of existing plans., *Artifical Intelligence*, Vol. 127, pp 199–220.

Machado, R. & Bordini, R.H. (2002), Running AgentSpeak(L) agents on SIMAGENT *in* Meyer J. & Tambe, M., eds.,'pre-proceedings of the 8th International Workshop on Agent Theories, Architectures and Languages', LNCS Vol. 2333, Springer Verlag: Berlin, Germany.

Lin, Z.-N., Hsu, H.-J. & Wang, F.-J. (2005), Intention Scheduling for BDI agents, *in* 'International Conference on Infomration Technology: Coding and Computing',ITCC-05, Vol-II.

Thangarajah, J., Padhgam, L. & Winikoff, M. (2003), Detecting and Avoiding Interference Between Goals in Intelligent Agents, *in* 'Proceedings of the 18th International Joint Conference on Artificial Intelligence',IJCAI 2003, pp. 721–726, Acapulco, Maxcio.

Thangarajah, J. (2004), Managing the Concurrent Execution of Goals in Intelligent Agents, Ph.D., RMIT, Australia.

Chalmers, S. & Gray, P.M.D. (2001), BDI agents and constraint logic, *AISB Journal Special Issue on Agent Technology* Vol. 1, pp. 21–40.

Dasgupta, A. & Ghose, A.K. (2005),Dealing with Objectives in a Constraint-Based Extension to AgentSpeak(L), *in* 'Eighth Pacific Rim Workshop on Multi-Agent Systems', PRIMA 2005.