2008

Online Collaborative Time Management System using Artificial Intelligence

Anand Sivaramakrishnan
San Jose State University

CS298 Report

Online Collaborative Time Management System using Artificial Intelligence.

Anand Sivaramakrishnan

anandsk123@gmail.com

Advisor: Dr. Chris Pollett

Department of Computer Science

San Jose State University

Fall 2008

Anand Sivaramakrishnan

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Chris Pollett

Dr. Robert Chun

Dr. Teng Moh

Abstract

Online Collaborative Time Management System is a system that will plan events and help achieve goals intelligently using Partial Order Planning. This product is similar to a social networking site, which plans events for collaborative set of people. For this masters' project, such a system is developed. Our site allows multiple people to add items to a collaborative To-Do list. The user interface for our site allows people to add actions and preconditions to the existing system, enabling the system to grow constantly. In order to generate a sequence of actions to the most complex problems, partial order generators are implemented at the back end. The planning algorithms are designed such that they will resolve conflicts by implementing backtracking whenever it comes across a conflict.

TABLE OF CONTENTS

LIST OF FIGURES

1. Introduction

Online Collaborative Time Management System project implements an intelligent online collaborative system for scheduling different groups' TO DO lists according to their daily routine. This system will differ from all other existing planners such as Google, Planware and GradeMate, as it does not stress on planning with respect to time, but will stress on planning logically to solve complex problems and tasks. It is collaborative in nature to enable planning between groups of people. Partial Order Planning algorithm is the backbone behind successful implementation of the system. The most unique feature about this system is that it is dynamic in nature, as it will be constantly growing.

The problem is that currently there is no tool available online to make plans for the most complex situations, which needs intelligence. For any event or goal, there are many important issues to deal with, and it is not possible to remember all of them. Furthermore, it is difficult to solve all the issues intelligently. The existing online planners have solved the first problem of remembering all the remaining issues to be solved. A system that helps solving all these issues intelligently is not yet available online.

My goal was to solve the above problem, and make the system accessible to many people where they can share problems and solutions.

After the successful implementation of the above stated goals, this report further explains the way the product is to be used (manual), how the backend planning algorithm works, and why this planner can be reliable.

2. Building a Planner

The primary goal of my project was to build a planner that will generate the plans for group specific goals and present it to the people who want their problems (goal) solved or events (goal) planned.

A planner operates by searching through a world of states. These states are fundamentally situations, which are the possibilities while travelling from the current state to the goal state. The initial state is the problem state. This is why it is called **situation space** planner, as it is going through all the situations. Every node represents a state/situation. There are two types of planners. Progression Planner searches forward from the initial state (situation) to the goal state (situation). Regression Planner searches backward from the goal state (situation) to the initial state (situation). Progression Planner searches forward from the initial state (situation) to the goal state (situation). Regression Planner searches backward from the goal state (situation) to the initial state (situation). In my project, I will be using backward search techniques, which means I will be traversing from the goal state to the initial state to get the sequence of actions.

During the CS297 implementation of the back end, I used STRIPS language to represent the initial state, the goal state, actions and preconditions.

During the CS 298 representations the object property value triples were used as inputs to the system, whereas the planner reads it as predicate wise collection of data, where each predicate is a collection of values for one distinct object.

3. Total Order Planning

In the project, we tried to make use of partial order planning algorithm. A Partial Order Planner is an extension of a Total Order Planner. This is why in the very first deliverable in CS297; I implemented the total order planner. Total order planning refers to a strict sequence of steps that has to be followed in order to obtain a solution for a problem or a query. It is a planner, which is inflexible and rigid in nature. It generates a plan to solve a problem, which may not be the best solution to solve the problem. Total order planner is also called linear planner, because of its narrow choice of steps to be followed to solve the problem in hand. This is a single solution from beginning to end, where in there are specified places where we have to enter data. In my implementation, I have implemented the following algorithm for my backend:

TO (P, G)

1. Termination check: If G is empty, report success and stop.
2. Goal selection: Let c be a goal in G, and let Oneed be the plan step for which c is a precondition.
3. Operator selection: Let Oadd be an operator in the library that adds c.

   If there is no such Oadd, then terminate and report failure.

   Backtracking point - all such operators must be considered.
4. Ordering selection: Let Odel be the last deleter of c.

   Insert Oadd somewhere between Odel and Oneed, call the resulting plan P'.
5. Update goal set: Let G' be the set of preconditions in P' that are not true.
6. Recursive invocation: TO(P',G') [7]

Figure 1: Algorithm - Total Order Planning

The input to the above algorithm is a text file, which contains information such as the initial state, which is the actual state in which the person trying to achieve a goal is actually at. He needs to reach the goal state from this state. The goal state is specified in the text file. The text file also has the total number of actions that are available to choose from when we decide how to traverse from the initial state to the goal state.

When I run the above algorithm with the specified text file as the input to the same, it will generate the plan. There will be just one plan that will be generated. Following this plan will take you from the initial state to the goal state. This plan takes into account all the preconditions for all the actions that are involved in the sequence of actions. This means that it will take into account all the constraints that could come in the way realistically whilst traversing from the initial state to the goal state.

When we implement such planning algorithms, the program written also needs to implement the backtracking function, which when called resolves conflicts that arises when we design the sequence of actions in the plan. This means during the traversal of different states, when we find it is not possible to implement the next action because of its precondition that cannot be satisfied. At such a stage, we need to go back and change one of the actions that we had chosen and choose a different action that has the same effect.

4. Partial Order Planning

In the second deliverable of CS297, I implemented a partial order planner, which was finally used in the Website. The partial-order planner is a planner that generates a solution, which is a set of open conditions that is empty, and there is no conflict. A planner that can represent plans in which some steps are ordered (before or after) with respect to each other and other steps are unordered.

Since partial numbers of steps are ordered, it is called Partial Order Planner. It is said that the Partial Order Planner follows the principle of Least Commitment, which says, 'One should make choices only about things that you currently care about, leaving the others to be worked out later'. Mountain Climbing is analogous to this principle.

UA(P,G)

1. Termination check: If G is empty, report success and stop.

2. Goal selection: Let c be a goal in G, and let Oneed be the plan step for which c is a precondition.

3. Operator selection: Let Oadd be an operator in the library that adds c.

If there is no such Oadd, then terminate and report failure.

Backtracking point - all such operators must be considered.

4. Ordering selection: Let Odel be the last deleter of c.

Order Oadd after Odel and before Oneed

Repeat until there are no interactions:

- Select a step Oint that interacts with Oadd.
- Order Oint either before or after Oadd.

Backtracking point - both orders must be considered for completeness.

b Let P' be the resulting plan.

5. Update goal set: Let G' be the set of preconditions in P' that are not true.

6. Recursive invocation: TO(P',G') [7]

Figure 2: Algorithm - Partial Order Planning

As implemented in the Total Order Planning Algorithm, the input to the above Partial Order Planning algorithm is again a text file, which has the format of actions, preconditions and effects as the earlier text files of the Total Order Planning Algorithm. The implementation of the Partial Order Algorithm was an extension of the Total Order Planning Algorithm. It had to implement Partial Orders. The generation of partial orders is done by a separate function that is called by this same algorithm when it is generating the sequence of actions. This separate function generates partial orders, which will tell

the user of this planner, the sections of the actions that it can implement in parallel and other sections, which have to be implemented strictly sequentially. The below figure demonstrates the pictorial behavior of the above planners. The Total Order Planner will generate only one of the two outputs that are shown in the figure. The goal that is being planned in the figure below is that of replacing a flat tire on the axle with the spare tire, which is on the trunk.

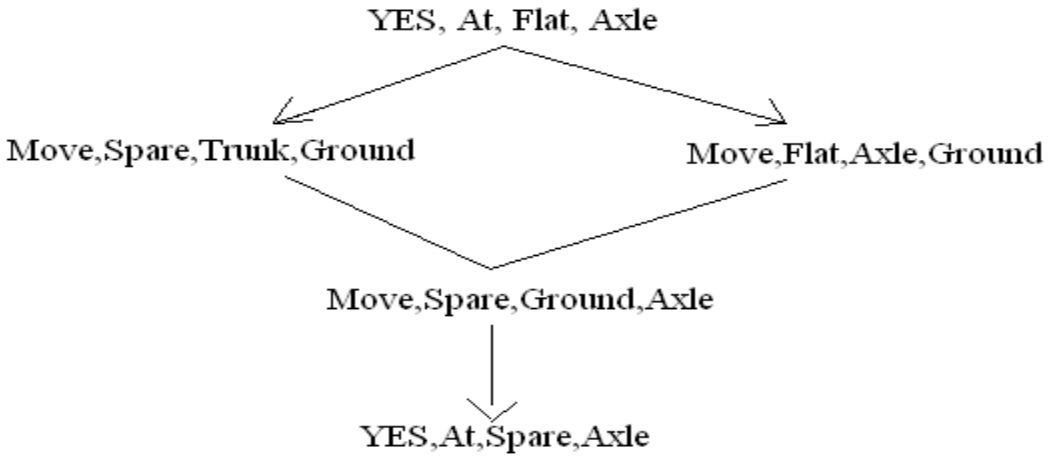

Figure 3: Total Planner and Partial Order Planner outputs for the goal of replacing flat tire on the axle

In the third deliverable of CS 297, I converted the above coded Partial Order Planner in Object Oriented Programming by making a class outside the working functions.

Object Oriented Programming is the type of programming where we convert all the code into an object. This means that we are combining both the state and behavior into an object. Therefore, now we have packaged an object consisting of data as well as the properties. That is the beauty of Object Oriented Programming.

During the build of the actual product the following approach was used to design a plan. In our approach, the algorithm is designed such that it will accommodate collaborative planning. This is such, because, it picks up options only within the group. This includes items, properties, actions and goals.

The design pattern of the algorithm is such that there is a class which has distinct functions to perform.

First, there is a collectInfo() function that collects all the goals that a user is associated with. This means that all groups that a user is associated with will have goals. The above list of goals is the cumulative list of goals that will be presented to the user.

For each of the above goals, the planner is called to get a plan to execute the goal. The user can further click on the goal (button), which will take him to another page where he will be shown all the actions of the plan that are assigned to him. He can further reassign few or all of these actions to other members of the group. There will also be a check box ahead of all these actions to check that ToDo if it is already done.

The planner method takes as input the Goal and the Group with which that goal is associated and further tries to design a plan with the available actions in that group.

There are three methods that this planner is further associated with. First of them is the unification method that checks to see whether a single predicate of the effect of an action matches a single predicate of the precondition of an action that is already chosen. If it does find an action that unifies, it will further call another function unificationDBUpdate() which will check whether all the effects of that action unifies with the world state. That is the action in synchronization with the world state. If this is satisfied, then this function will update the world state. If there is an instance where none of the actions satisfy a precondition then the planner calls the third and last of the functions that it is associated with which is the backtrack() function. This function will backtrack to the last action to affect the world state, cut the generated plan and restrict it to this action excluding it. The backtrack() function will further look out for alternate actions that will satisfy the preconditions of the action just before the conflict action in the backward heuristic search that was generated.

The speed of the algorithm is slow in general. Specifically, it is as follows:

For each plan:

(No. of predicates associated with the plan) * (No. of properties for each predicate respectively)

The above is true because, for each action to satisfy a precondition we unify all the properties of each predicate that the action is associated with the world state during the unificationDBUpdate() function call.

In fact this algorithm can be frustratingly slow when it is generating large plans.

If there is still no plan generated the user is prompted to add more actions (to-do) to the group and then re-visit this page.

5. Website Design and Goal

There was a lot of effort being put into the design of this website so that the users are required minimum use of the keyboard and maximum use of the mouse. The functionality of this website is also simple to understand. Whilst just playing around with the links for five minutes, the user will understand how easy it is to use, and he/she will also realize the benefits in terms of time saving and accurate planning is magnanimous. The site as its motive is, is completely group oriented and focuses on collaboration. Users of this site can do the following things:

1) Create User Profile – Registration

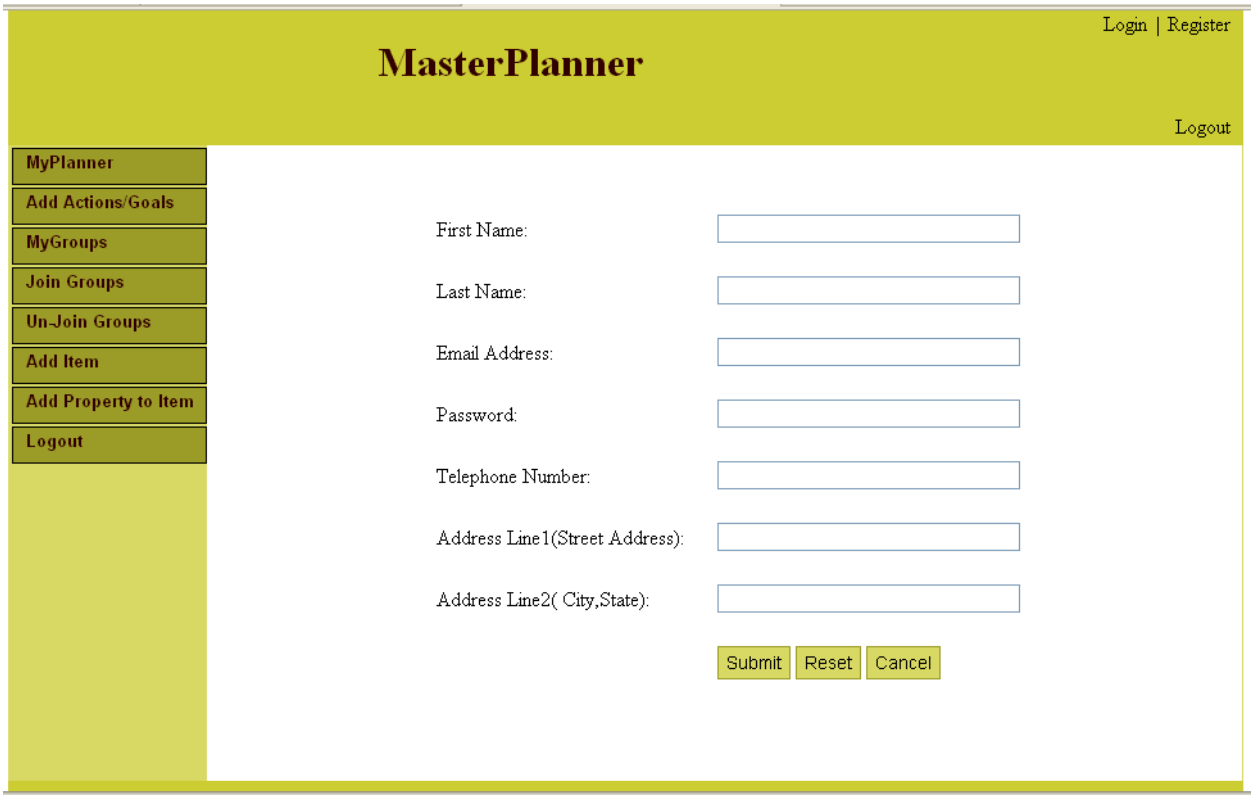

Figure 4: New User Registration

2) Login – Using Login and Password

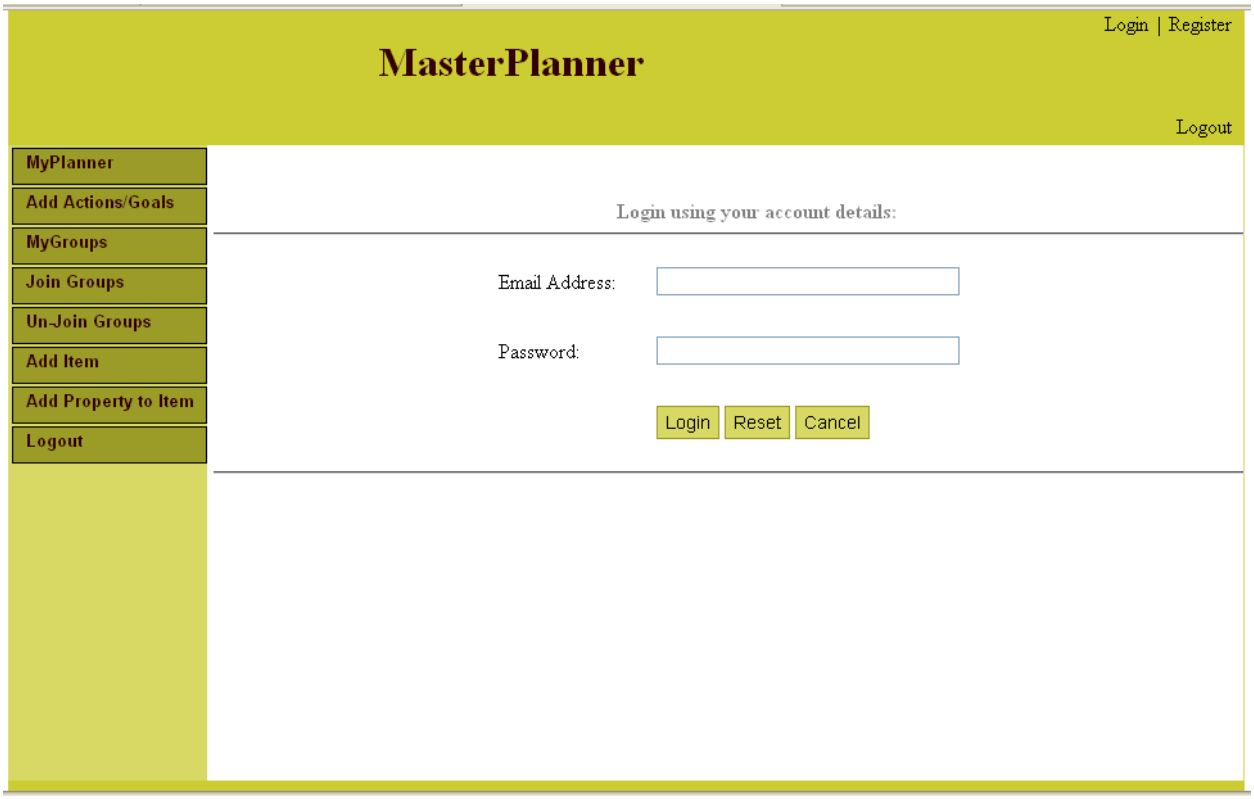

Figure 5: Login Page

User has to login at least once before he/she has to start using the planner. After the logins he could be logged in next time automatically with cookies set in the client side.

3) Create Groups – Collaboration

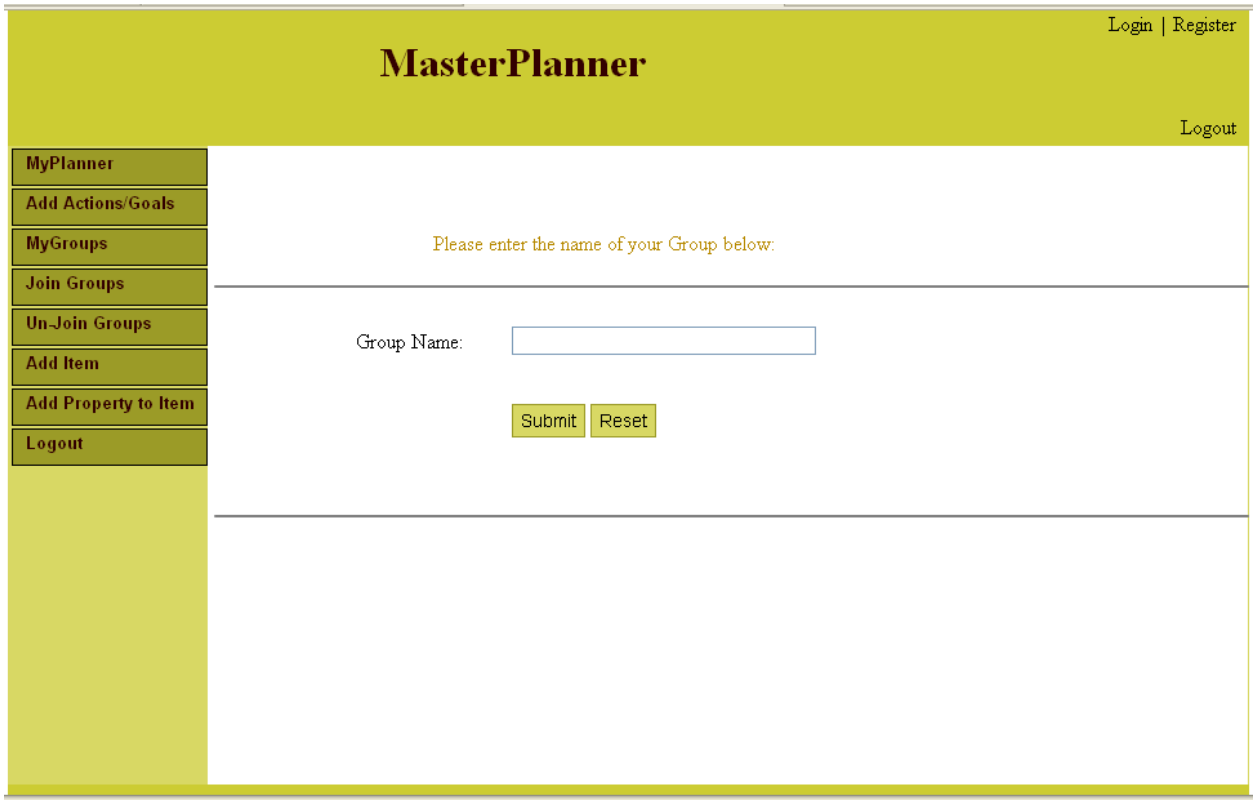

Figure 6: Create New Group

For categorization of goals as well as people, user creates goal type groups. Therefore, one group can have similar groups. Plus, if a user wants to be a part of multiple groups he can create or join groups that are different than he is already a part of.

For example, a user can be a part of one group that is between him and his roommates and also be a part of another group consisting of his colleagues in office. Therefore, he can view goals based on categorization of location or circumstance.

For instance, a person can have the following goals between him and his roommates:

1) Party

2) Buy Groceries

And can have another set of goal between him and his colleagues:

1) Project A

2) Project B

4) Create Actions, along with its' Preconditions and Effects – To Do.

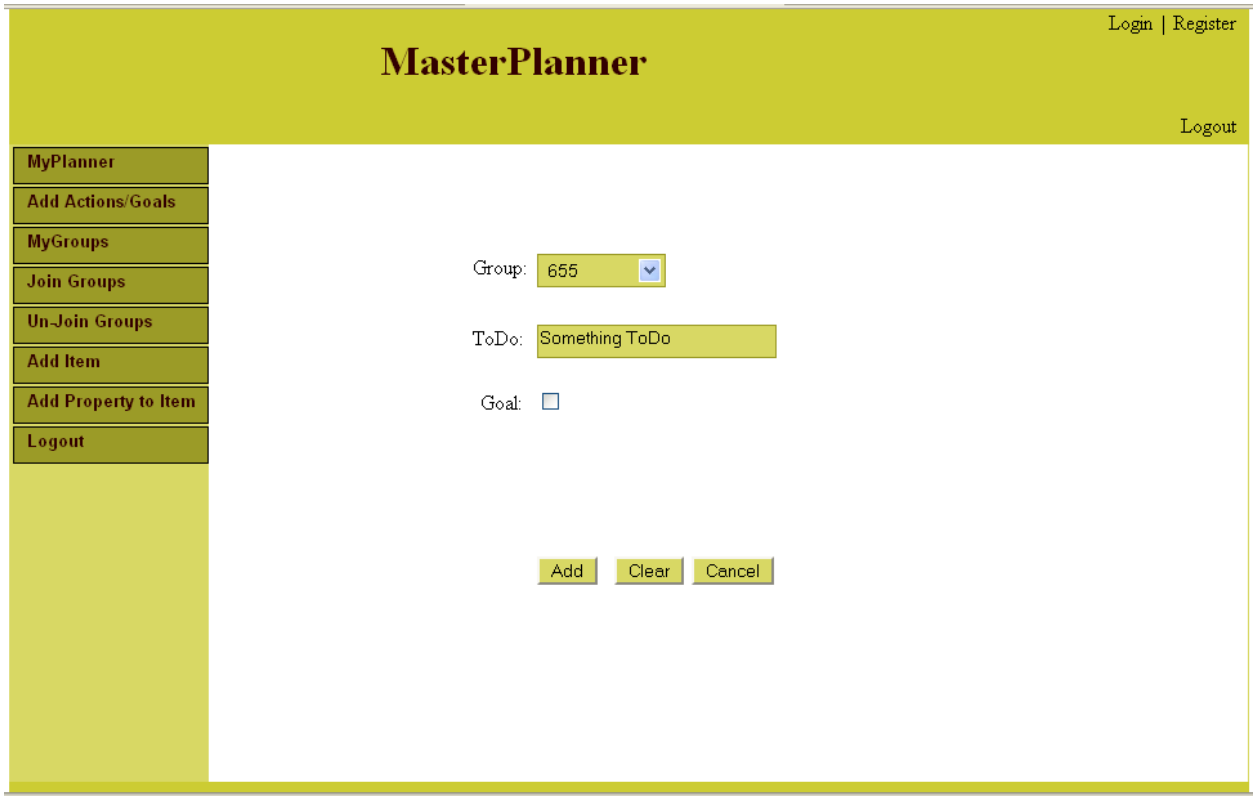

Figure 7: Adding actions to a Group

The actions that are added to the group are like ladders used to reach the goal. Each action can further be added with preconditions and effects, to tell the planner that to

use. This action certain preconditions are supposed to be satisfied in the world state. Also there are effects of these actions on the world state.

Figure 8: Adding Preconditions and Effects after adding an Action

Every precondition and effect of an action/goal is a form of a triple, item, property and value. Now, it is this same way that it is stored in the database. But, when the planner fetches this data it clubs, all object specific data to form predicates. It is using these predicates that the planner traverses from the goal to the initial state (backward heuristic search). The initial state is defined by actions with no preconditions.

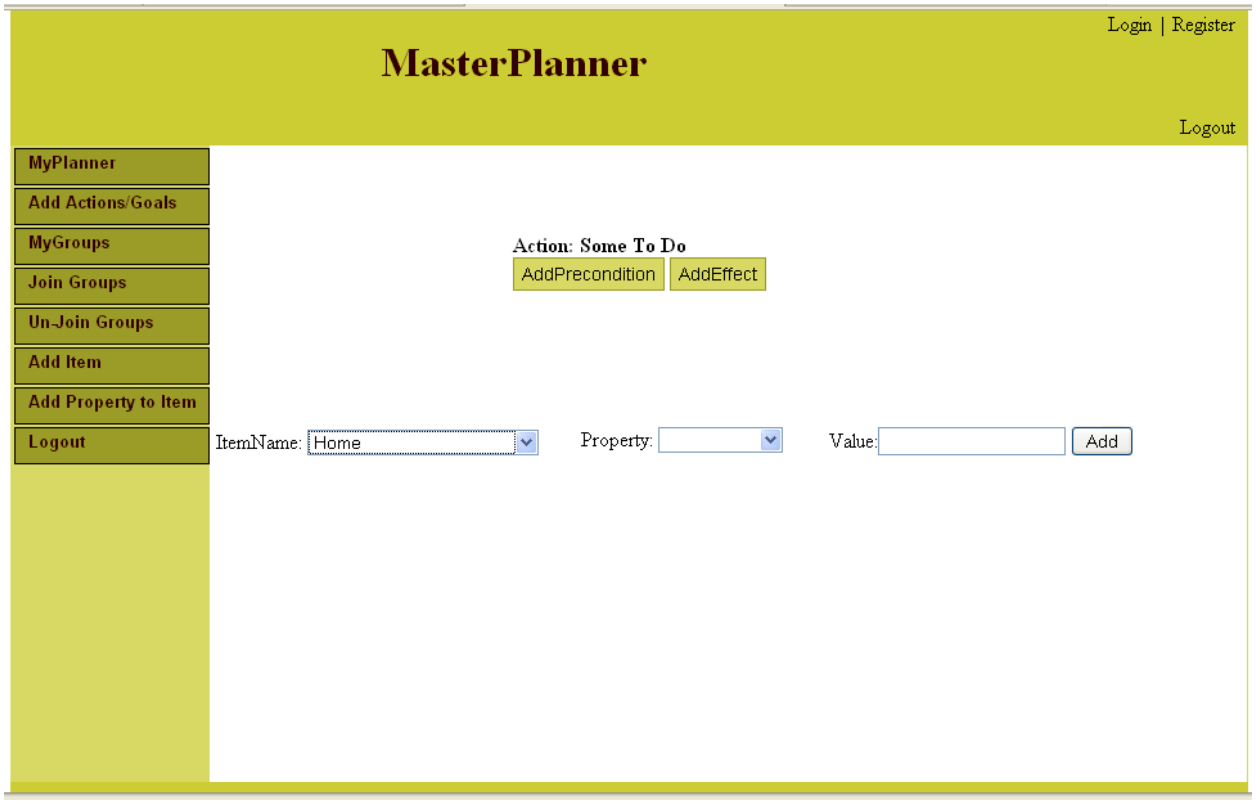

Figure 9: Actual Addition of the Triples (for Preconditions and Effects)

5) Create Goals (synonymous to Events)

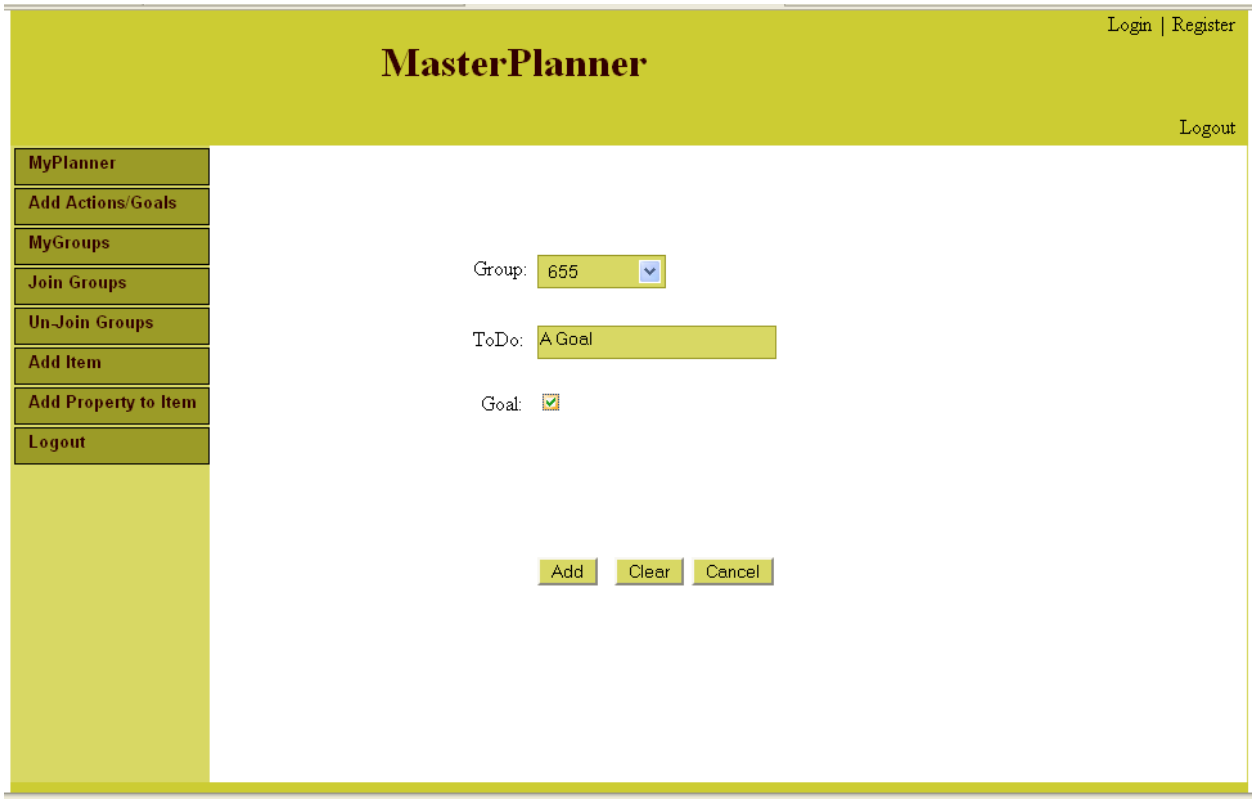

Figure 10: Add Goals

This is the core motive of the planner. Making plans for specific goals. Generally, groups have goals of the same subject with slight twists. Every goal can have a plan only if it has preconditions. Else, it has no plan.

For example, the following Goal can be added to the group:

Party

Precondition:

1) Home Clean Complete

2) Home Food Ready

6) **Join Groups**

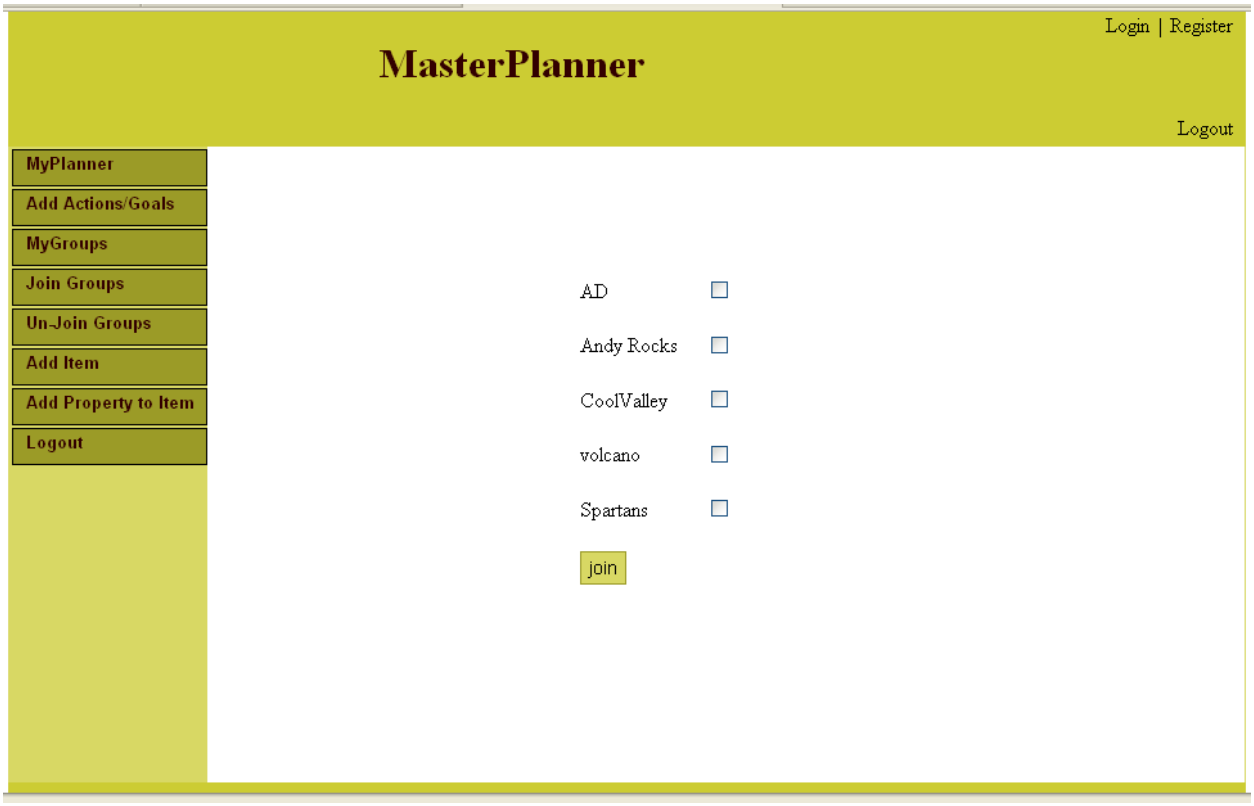

Figure 11: Join Groups

User is given option of groups that are open. In such a manner he will be able to collaborate with groups that have the similar motive as he has but he need not necessarily own it. Therefore, he will have many ideas and techniques of goals and actions of fields that he is interested in. He can view solutions to problems that he exactly did not know what/how the solution would be. As and when many people use

this planner, this system will be a resource for knowledge, ideas and techniques for subjects that groups are already formed in.

7) Un-join Groups

Figure 12: Unjoin Groups

As mentioned before, there is flexibility in this system. There is no commitment to stay in a group for a specific or indefinite period of time. But even though a person leaves the group he/she has to fulfill the tasks that are assigned to him/her.

8) Add Item

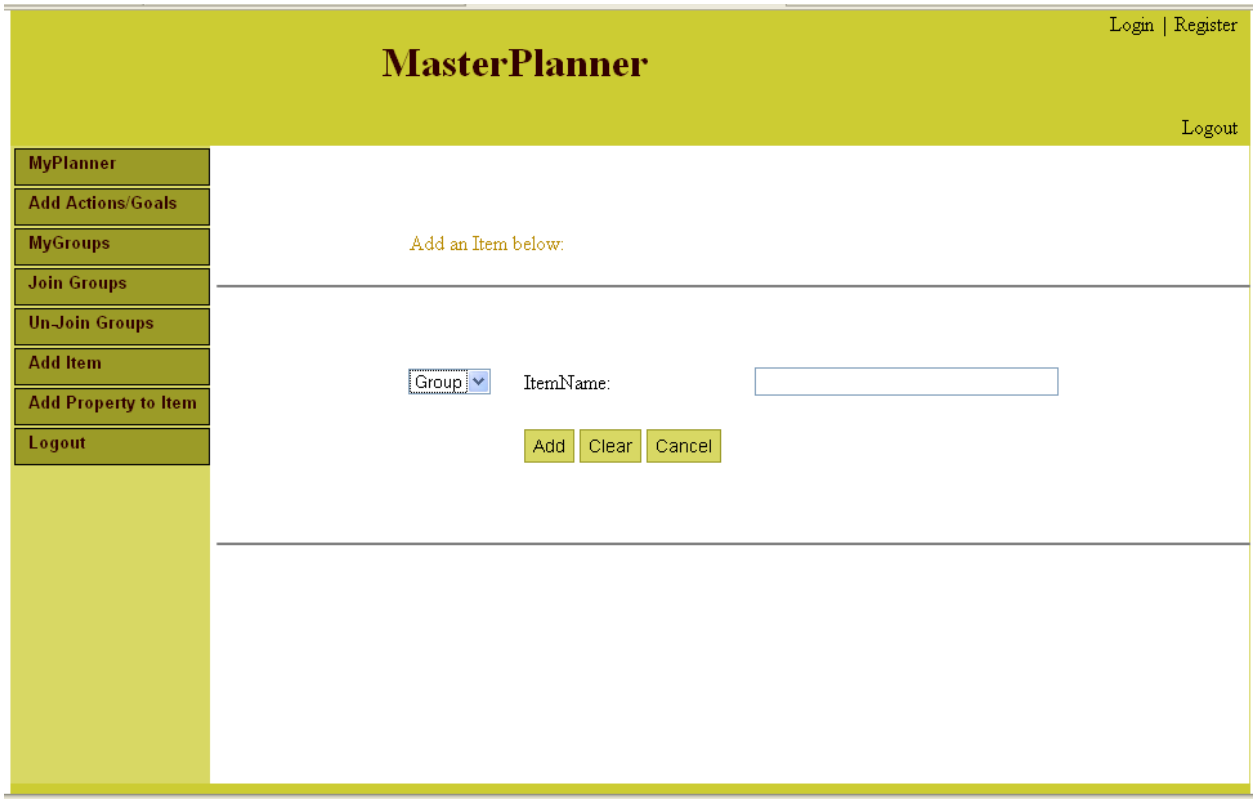

Figure 13: Add Item

Adding items to the database is again Group specific. After adding an item the group member is given an option of adding up to 5 properties to the item. If you want to add more than 5 properties to the system, it has to be done explicitly using the next tab.

For example, in a group we can the following objects:

1) Home

2) Car

3) Office

9) Add Property to Item

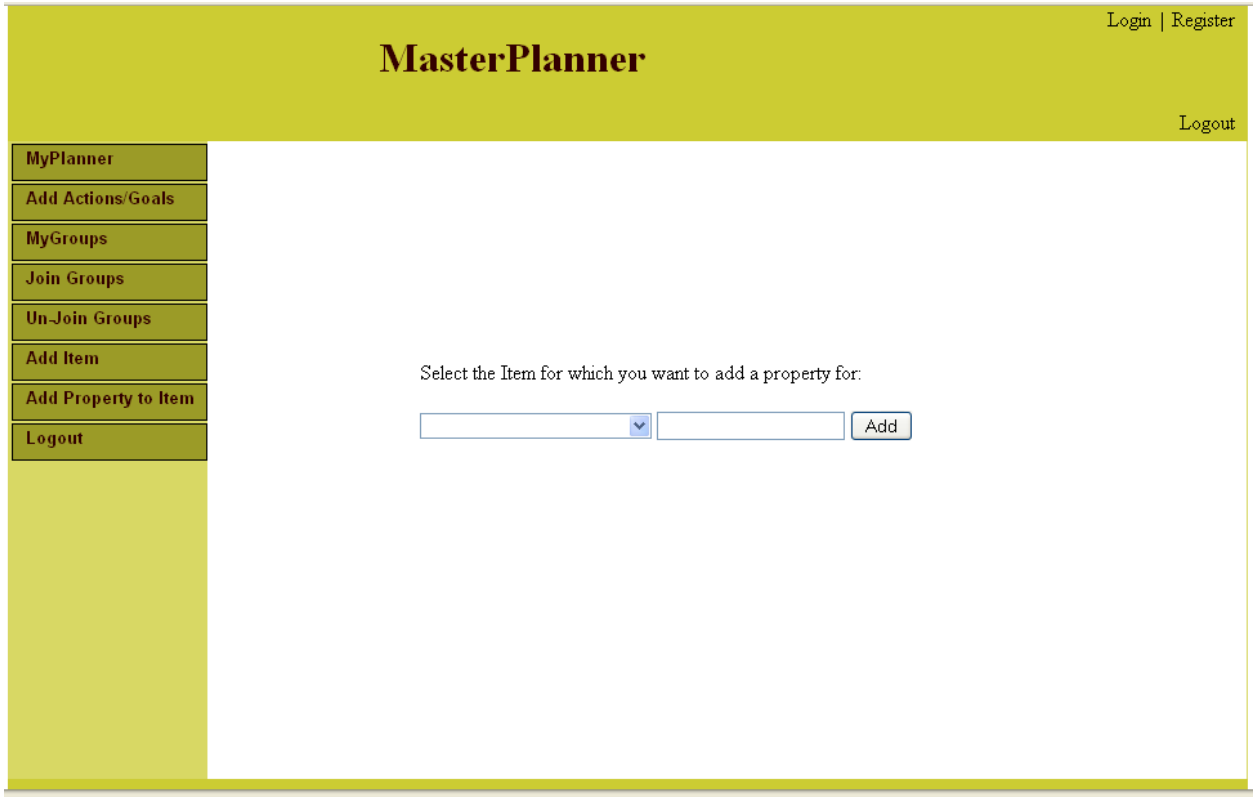

Figure 14: Add Property to Item

Adding different properties to the added items is one of the activities that the group members can do. This is increase the scope of the addition of preconditions and actions, making goals a more complicated and wider task.

For example, if 'HOME' is an item in a group. There can be the following properties in it:

1) Cleanliness

2) Food

The values that can be added to the Cleanliness property can be added on the fly during adding preconditions and effects. There is no fixed range for the values. Values are interpreted as strings by the planner.

10) View ToDos and Assign Task to Team Member

Figure 10: Home Page, after logging in

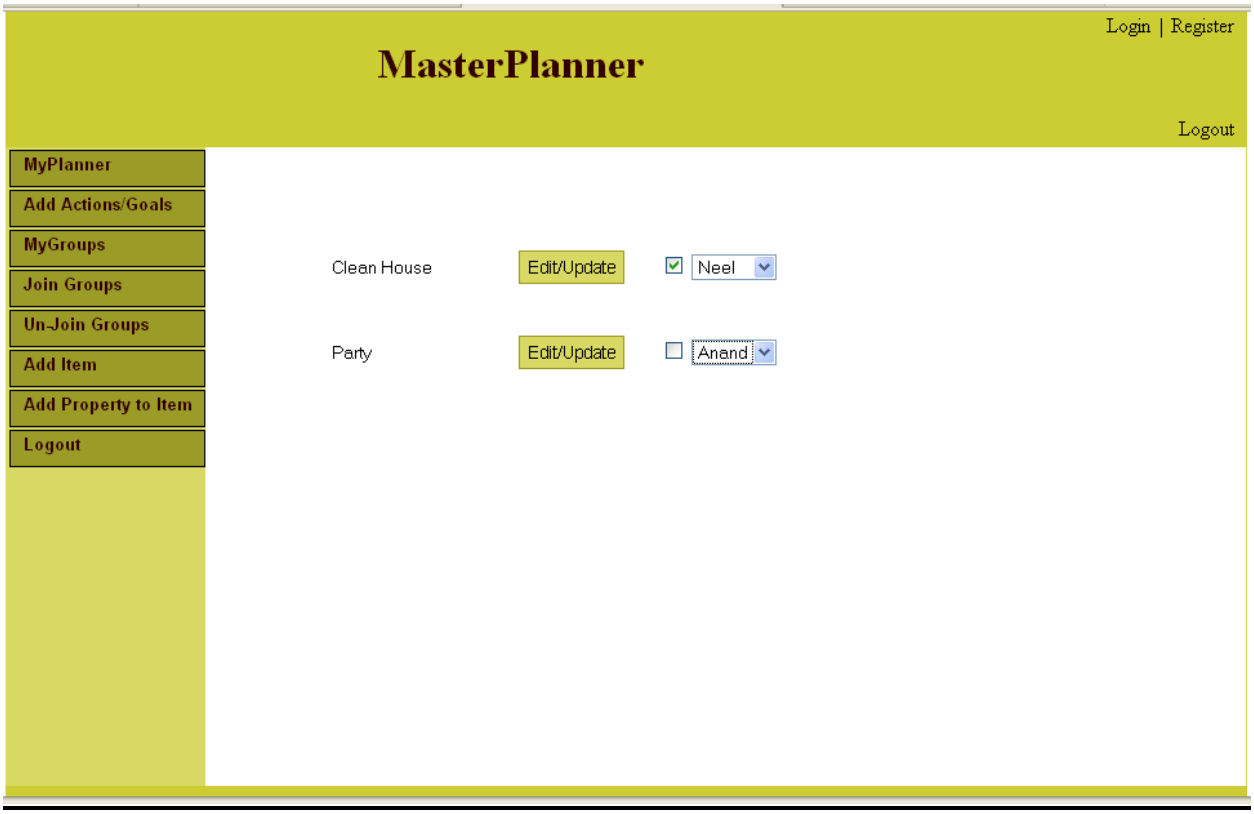

Figure 11: View ToDos and Assign Task to Team Member

The last but not the least activity that a user can do is view the actions that he has to do for each goal. Also he can assign any of these actions to any other group member, from the group which the goal belongs to.

This will help in appropriate sharing of work. Also, even if one of the group-members un-joins a group he still has the responsibility of performing the task allocated to him. In case he un-joins he does not have the option of transferring the task to anyone else.

This page will only display actions that are currently the logged in user's actions responsibility.

6. WEBSITE

The product that is being developed is a web site. It has registration and all the other details. It is the main interactive interface of the customer (groups). The customer (groups) feeds in actions/ goals, their preconditions and effects. They can also create groups, and assign actions / to-do to other members of the group. The normal sequence of actions that the user will perform will be based on the preconditions and effects that he or she enters for actions and goals (only preconditions for goals). The planning algorithm will be involved on a per goal basis, when the sequence of steps, are displayed to the user. The planning algorithm is based on a partial order planning as mentioned above. Creation of objects, properties and values are on a per user basis, for groups.

The Website has been mainly developed using PHP at the backend, and HTML, Javascript and CSS at the front end. Also there has been use of AJAX for dynamic changes by the user at the front end without page refreshes.

(a) PHP – PHP (Personal Home Page) Hypertext Processor

All of the server side programming has been done in PHP. PHP is one of the top 5 programming languages in the world, as it is one of the most powerful server side scripting languages. The Partial Order Planning algorithm has also been implemented in PHP. It is implemented using the Apache Web Server. It is open source. You can download it at http://www.php.net/

(b) CSS – Cascading Style Sheet

CSS has been used to determine the look and feel of the web site. The color, positioning, and button variety has been introduced by the same. It is rendered by the browser.

(c)JS – Javascript

JS has been used for purposes like validation, and on change functionalities. It is also rendered by the browser.

(d)AJAX – Asynchronous Javascript and XML

AJAX is mainly used to create faster, better and user friendly web applications. AJAX makes the whole experience of the user using the website a cool and fun experience, without too many annoying page refreshes. A combination of HTTP request and Javascript is what makes AJAX.

The front end will enable the users to interact with the system effectively. The users can add their actions and goals to the system to make the system a growing collaborative planner to work with, so that it can build convenient plans, and traverse solutions for complex problems. However, for the users to be willing to give input to the system, the system must have a very attractive user interface. It has to have minimum typing required to give the input. It should have many checkboxes and drop down menus.As this system is a collaborative system, it needs to show interactions between different

people of the same group. To design the user interface for a collaborative system is difficult and a specimen for such design is helpful.

As mentioned the GUI has been made in HTML (Hyper Text Markup Language), CSS (Cascading Style Sheet), and Javascript, and also AJAX.

7. Implementation

The code has been written such that it is well abstracted as to which folder contains what content functionally. The file names, variable names and function names are as descriptive as possible. It is arranged in the following way:

There are the following sub folders inside the main folder:

js – all Javascript files are into the same folder named JS.

css – all cascading style sheet files go into this folder. The look and feel of the website, are rendered by files in this folder.

images - all the image png files are in this folder.

config – this folder contains the configuration files such as what database to connect to with what username and password

Apart from the above subfolders, the main folder also has a bunch of php files. The important ones are mentioned below:

1) planner.php – contains the planning algorithm mentioned above in section 4
2) backtrack.php – contains the backtracking function
3) addItem.php – adding group specific items into the database.
4) addToDo.php – adding an action along with its preconditions and effects

There are many such function specific files, like assigning a task of a plan to someone else in the group.

Also, generally in artificial intelligence, predicates in the system are classes. In our system, the items entered into the system are nothing but objects.

This means there is a one to one mapping between classes (blueprint of objects) and actual objects. Every entry of an item into a database is an entry of both an object and a class. Predicates are generally associated with classes, but in this case it is associated with both the class as well as the object.

When a group enters an item, they are expected to enter an object instead of a class. In lay man terms they are expected to enter proper nouns (objects) instead of common nouns (classes).

Below are a few examples:

1) Members of group are supposed to enter item name “Chocolate house” or “Peter’s Houses” instead of “House”.

2) Members of group are supposed to enter item name “Yellow Car” or “Red Car” instead of “Car”.

Therefore, when they are entering preconditions and effects they are specifying object property value triples to the system, which are further clubbed by the algorithm as predicates. Each predicate is a summation of the object-property-value triples in to just

one predicate with number of arguments as the number of properties holding the values specified for the properties.

The planning algorithm is the same as discussed above under Partial Order Planning section. As mentioned, in the case of the product built in CS298 the unification algorithm has been used for matching preconditions and effects while traversing from the goal to the initial state in the backward heuristic search. In designing the plans we have to maintain consistency between the effects of the state we choose and the world state. The database has a table named World State used for the maintenance of the current world state during the run of the planner. This table is generally empty it is used only during the brief run of the algorithm which could be prolonged during the creation of long plans.

The effects of every action chosen, has to be synchronized with the world states in every predicate that exists in the world state. Therefore, it has to be carefully checked for inconsistency. Also, once the action is applied after checking for inconsistency, the world state is flushed such that the preconditions of the chosen action can occupy the table for the respective predicates. During the satisfaction of one of the preconditions, if the planner does not find any action that does the job then it will back track the latest action that affected the world state. After deleting this id from the plan we find out whether there is an alternative for satisfying the previous action's preconditions in the reverse arrangement of the plan TO-DO ids.

Creation of plans is partial responsibility of the user as he/she has to add sufficient actions corresponding to their goals, for the partial order planning algorithm to generate a logical set of actions for the user to reach the goal.

For example,

When the user enters the following 2 preconditions

1) Home – Cleanliness – “Complete”

and

2) Home – Food – “Ready”

for the Action/Goal “Party”

It is interpreted by the planner at the back end as

Home (Cleanliness, Food)

as

Home (“Complete”, “Ready”)

The unification is done as follows:

Action: Clean House

Effect: Home – Cleanliness – “Complete”

Above effect to the planner is, Home (“Complete”, Food).

Unification with Home (“Complete”, “Ready”)……..World State

Therefore, after unification, world state will be as follows:

Home (Cleanliness, “Ready”)

Action: Prepare Food

Effect: Home – Food – “Ready”

Above effect to the planner is, Home (Cleanliness, “Ready”).

Unification with Home (Cleanliness, “Ready”)……..World State

Therefore, after unification, world state will be as follows:

Home (Cleanliness, Food)

Hence the world state is being affected with both the actions, thus satisfying both the preconditions. Further, if there are preconditions of the above mentioned actions the world state will be flushed out and fresh preconditions (predicate values) will be added into the world state.

8. Database Structure

The Database is most importantly used to store preconditions and effects of actions. Also the database stores group specific actions. That is every action can only be used by the group it belongs to.

MySQL has been used for the implementation and set up of the database. MySQL is again open source and can be downloaded from

http://dev.mysql.com/downloads/

MySQL can be easily connected with code written is most of the programming languages. In this case it has been connected to PHP (code-wise).

The database has the following tables:

1) Users - The User table basically has the user registered information in it. It has the following fields:

   (a) UserId – Primary Key

   (b) FirstName - of the user.

   (c) LastName - of the user.

   (d) Address - of the user.

   (e) Email - of the user.

   (f) Telephone – of the user.

(g) Password – of the user.

2) Todo – Actions and Goals that the users want to perform and achieve respectively. This table has information stored using the following fields:

   (a) ToDoId – Primary Key

   (b) GroupId – The group that the todo belongs to. Foreign Key.

   (c) GroupId – Id of the group that this action/goal belongs to.

   (d) ToDoDesc – The text description of the action, which will be displayed to the user in this tuple.

   (e) UserId – The user who either created and/or assigned to perform this action.

   (f) Goal – Boolean field, will be check only if this is the goal.

3) Preconditions - of actions and goals stored in he todo table. This table has the following fields:

   (a) PreconditionId – Primary Key

   (b) ToDoId – the action/goal, for which this is a precondition.

   (c) ObjectId – the object that is what this precondition is associated with.

   (d) PropertyId – the property of the above object that is what this precondition is associated with.

   (e) Value – the value of the object property.

4) Effects - of actions and goals stored in the todo table. This table has the following fields:

(a) EffectId – Primary Key

(b) ToDoId – the action/goal, for which this is a effect.

(c) ObjectId – the object that is what this effect is associated with.

(d) PropertyId – the property of the above object that is what this effect is associated with.

(e) Value – the value of the object property

WorldState – Reserve Database, used to detect database. Generally, this database has to no data. It has data for very short period of time when the planner generating plans. It is generally empty. This table stores the predicates and checks whether they are achieved from goal state to the initial state.

It has the following fields:

(a) ToDoId – the action/goal, which last affected the world state.

(b) ObjectId – the object that is what this precondition is associated with.

(c) PropertyId – the property of the above object that is what this precondition is associated with.

(d) Value – the value of the object property

5) Groups – the group details of every user. It has the following fields:

   (a) GroupId – Group Identifier

   (b) UserId – user associated with the group.

   (c) OwnerId – user that owns the group.

   (d) GroupName – name of the group.

6) Objects – associated with a Group. It has the following fields:

   (a) ObjectId – Primary Key

   (b) ObjectName – name of the object.

   (c) GroupId – Group that owns the object.

7) Properties

   (a) PropertyId – Primary Key

   (b) ObjectId – Object with which the property is associated with.

   (c) PropertyName – name of the property.

9. Testing & Experience

As and when different test cases were applied to the system it was understood that the input to the unification had to be all the properties, of each predicate that existed in the world state. If it was not mentioned in the precondition or effects of actions/ goals then it would be a variable which is the property name. Every action applied has its effects affect the world state accordingly for whichever predicate it affects. At the end of the run of the algorithm when we observe the world state, its predicate will be in sync with the last action that affected the respective predicate.

One of the scenarios is as follows:

Goal: Party

Preconditions:

1) House Cleanliness Complete
2) House Food Ready

Available Actions:

(A) Clean House

Effect: 1) House Cleanliness Complete

(B) Make Food

Effect: 1) House Cleanliness Incomplete

2) House Food Ready

(C) Make Food Neatly

Effect: 1) House Cleanliness Complete

2) House Food Ready

No preconditions for either of the above actions.

Initially, the planner will choose the following actions in the backward heuristic search:

Last: Party

Second Last: Make Food

When it realizes that Make Food will leave the cleanliness of the house incomplete again and it is just before the Party. It logically understands this through the world state table that the plan is going wrong. This is when it will backtrack to the previous action, that is party and find another action to make the food ready.

When it backtracks, it will find another action that completely unifies the preconditions of the party. Therefore, the final set of actions in the reverse order (backward heuristic search) is as follows:

Last: Party (Goal)

Second Last: Make Food Neatly

Third Last: Clean House

This plan was presented to the front user (of the system) in the following sequence:

1) Clean House

2) Make Food Neatly

3) Party (Goal)….Yeah!!!!!!!!!!!!!!

The above scenario should have given you a good sense of the planner solution.

When such a plan is presented to the user, the user is also given the choice of assigning any or many or all of the tasks to different group members of whom the goal belongs to.

Also, anybody in a group can un-join their group, anytime they wish. But they have to complete the tasks that are assigned to them, even though they have left the group. Once a person leaves the group he also looses the authority to assign the task given to him to any other member of the group.

When there is a failure to make a plan the member of a group can further add actions to a group, and observe if the planner still fails.

All members of a group are allowed to add actions (its preconditions and effects) and also goals to a group.

Another scenario is as follows

The plan is:

1) Switch On Lights
2) Make Food
3) Clean House
4) Party

In the above scenario the lights have to be switched on for all actions 2,3,4

One scenario is as follows

The plan is:

1) Switch On Lights
2) Make Food
3) Switch Off Lights
4) Change Clothes
5) Switch On Lights
6) Party

In the above scenario the lights have to be switched on for actions 2, and 6, where as it has to be switched off for action 4.

10. Usability Testing

Priyank Mohan: Initially, I had no clue how to use this product. The first 5 minutes was very frustrating. But then I started adding items and properties to the added items. After that I started adding some of the basic todos into the system like Sleep. This is when I realized that preconditions and effects are nothing but triples of object property and values inserted by the user dynamically.

The site is very simple. I am sure, by just playing around with it for just 5 minutes, people will know all its functionality. This site is a little less complicated, does not have the qualities of most of the commercial sites these days. But the features and functionality is more than acceptable. This product is a good base product to make plenty of enhancements in the future for a very sophisticated online planning site.

Dilip Menon:

To Do: The caption "ToDo" is confusing w.r.t a group. It would be easier to state something simpler like "Checklist Item" or something

Un-join groups – It would be easier to say "leave" group since un-join is a complicated word

Overall, the project is very useful for simple tasks within groups. It can be used as an event planner tool which I believe is the goal. The terminologies in the user interface (precondition, effect etc.) are a little cryptic and confusing at times. Hence this needs to be modified to be more user-friendly and easy-to-use.

The project does not work as a workflow tool since it has severe limitations such as:

- lack of sub-sections or groups, i.e. finance-team, accounts-team etc.

- lack of lookups on members of groups etc

- no dates or deadlines

So it cannot be used within a corporate network.

Response to above Feedback

1) The caption “Un-join” has been changed to “Leave”.
2) The terminologies ‘precondition’ and ‘effect’ will be changed to ‘after’ and ‘before’ respectively.
3) After discussion with my advisor I could pre add a few items and properties to group accounts by default.
4) There will be a help feature in the web site which will further help in the use of the system.

11. Conclusion

The implementation as in the proposal was group specific and collaborative. All objects, actions (to-do/rules), and goals are group specific, and can be entered by any member of the group. Therefore all actions available to the planner are added to the database by the group member. Insufficient actions available for a goal are reported as a failure and are reported to the group members (when logged in) as a plan failure, and they are prompted to add more actions before calling the planner for the same goal again.

There had to be changes made to the database structure multiple times for the algorithm to fetch data and update the world state in the database. Designing the database and getting the algorithm to work were the toughest and complex tasks. These 2 tasks were expected to be tough from the very beginning of the project. Decision making as to whether items added to the database were classes/objects were also very confusing. But, finally it was understood that these had to be objects. Each object is also a predicate.

The motive of learning how to build a system in AI (especially Planning) was achieved through this project. I was very curious at the beginning of the project.

Our planner is built such that it gives the user complete freedom and flexibility as to what to insert into the group details. The user can add any items and properties for items into the database.

Logical thinking and planning is a tedious and time consuming task, which this planner saves of its' users from doing.

12. References

1. Peter Norvig, Stuart Russell. (1995). Artificial Intelligence: A Modern Approach. Prentice Hall Series.

2. Sussanne Biundo, Maria Fox. (1999). Recent Advances in AI Planning. ECP, Springer.

3. Craig Knoblock, Qiang Yang. (1997). Relating the Performance of Partial Order Planning Algorithms to Domain Features. SIGART Bulletin, Vol. 6, No. 1, 8-15

4. Edwin P.D Pednault. (1988). Synthesizing plans that contain actions with context dependent effects. Computational Intelligence, 356-372

5. Stuart J. Russell. (1992). Efficient memory-bounded search algorithms. Proceedings of the Tenth European Conference

6. Charles. F. Schmidt

   Rutgers Institute, Notes on Planning

   http://www.rci.rutgers.edu/~cfs/472_html/Planning/PlanAlg_472.html