From theory to practice
The UTEP robot in the AAAI 96 and AAAI 97 robot contests

C. Baral, L. Floriano, A. Hardesty, D. Morales, M. Nogueira, and T.C. Son
Department of Computer Sc.
Univ. of Texas at El Paso
El Paso, TX, 79968, USA
{chitta,floriano,hardesty,dmorales,monica,tson}@cs.utep.edu

Abstract

In this paper[1] we describe the control aspects of Diablo, the UTEP mobile robot participant in two AAAI robot competitions. In the first competition, event one of the AAAI 96 robot contest, Diablo consistently scored 285[2] out of a total of 295 points. In the second competition, our robot won the first place in the event "Tidy Up" of the home vacuum contest. The main goal in this paper will be to show how the agent theories - based on action theories – developed at UTEP and by Saffioti et al. was used in the building of Diablo.

1 Introduction

We participated[3] in the AAAI 96 robot navigation contest [KNH97] and the AAAI 97 home vacuum contest. *In the first competition, our team scored 285 points in all runs of the contest out of a total of 295 points and was placed third in the finals. In the second competition, we won the first place in the event "Tidy Up."* In this paper we relate theory of agents, particularly the one developed at UTEP, for higher level control, and the one by Saffioti et al. [SKR95] for low level control, to the design of our contest entry, Diablo. In the process we present a top-down description of the control architecture of our robot and show how each level closely followed the theory. To focus on the main points, the relations will be made only with respect to the AAAI 96 competition. In the last section, we show how this architecture can be extended and modified for new tasks. We also discuss some of the weaknesses of our Diablo and present different goals which we would like to pursue.

In the AAAI 96 robot navigation contest [KNH97] robots were required to achieve a particular navigational task around a office like environment, given the topological map of the environment. In particular, the robot was required[4] to start from the directors office, find if conference room 1 was available (i.e., empty), if not then find if conference room 2 was empty. If either was empty then inform professor1, professor2 and the director about a meeting in that room, otherwise inform the professors and the director that the meeting would be at the directors office, and finally return to the directors office. Robots were required to do all this without hitting any obstacle, without

[1]**To Appear in Agent'98**
[2]We gave up 10 points in having a canned program to get out of the initial room. Otherwise our robot was perfect.
[3]We were also assisted by Alfredo Gabaldon, Richard Watson, Dara Morganstein and Glen Hutton.
[4]To focus on the main point we have simplified the real requirement a little bit.

changing the availability status of the conference rooms, and the robot was required to be back at the director's office.

The topological map of the office environment and our partitioning of it to landmarks is given in the following figure.

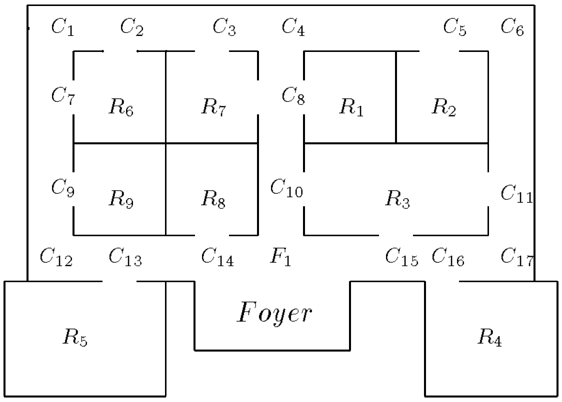

Figure 1: A Diagram Showing the Topological Map for Event 1 Participants at the AAAI 96 Robotics Competition.

2 Our theory of agents : a brief description

Our theory of agents is detailed in our earlier work [BS96, BS97a, BS97b]. In this theory we are concerned with the control of one agent in a dynamic environment, where exogenous events beyond the control of our agent may occur. Our agent has an action theory (such as a theory in the language [GL93, KL94, San94, BGP96]) where actions and their effects are specified together with constraints and causal rules about the environment.

Our theory contains a formulation of correctness of a control module – a control module is a collection of control rules of the form

if $p_1, \ldots, p_n$ **then** a,

where p_i's are fluents whose value is obtained through sensing and a is an action –

with respect to an action theory, a goal (could be achievement or maintainance goal), a set of plausible initial states, and a theory of occurance about the exogenous events. The formulation of correctness is based on two steps:
(i) First we compute the set of plausible states, called the *closure*, based on a given set of initial states, the control module and a theory about what exogenous events are likely to occur. (ii) Then, for each state in the closure, we *unfold* the control module by assuming that no exogenous events occur and test if the resulting plan (which may be a conditional plan [Lev96], if the state is incomplete and the control module has sensing actions) achieves the goal.
Note that, since exogenous events are taken care of in step (i), they do not need to be considered in step (ii).

We also develop sufficiency conditions for correctness that allows us to prove the correctness of a class of control modules, and more importantly they (the sufficiency conditions) give us an algorithm that automatically constructs control modules, given a goal, a set of initial states, and a theory about the exogenous actions.

Our approach to control agents is to take a goal, and an action theory about the agents actions and the environment, and either verify correctness of control rules provided by a control design person, or automatically construct a control module correct with respect to a set of given plausible states, or a combination of both. *All this is supposed to happen off-line.* (Unlike, universal plans [Sch87], we only consider a subset of all states, which are carefully constructed from a given set of plausible states based on what actions the agent is going to do and what exogenous events are highly likely. In [BS96], these set of states is referred to as the *closure*.) Our agent is supposed to be equipped with such a control module before it is on its own in the environment.

Once the agent is on-line, it is supposed to continuously sense and react based on its control module. In rare cases the agent may be in a state which was not accounted for off-line and there is no control rule prescribing what to do in such a case. If that happens, the agent deliberates, to reach one of the accounted for states. (Note that, since the set of accounted states is much larger than the set of states that satisfies the goal, it is much easier to make a plan to reach an accounted-for state.) From there the control module can tell the agent what to do. Note that our use of both deliberation and reaction is different from the traditional hybrid architectures [Ark91] (also in many of the papers in [Mae91]), where lower level modules are reactive and higher level modules are deliberative. In our case, the agent is reactive at all levels for the set of plausible states for which it is prepared, and it becomes deliberative when it encounters a rare state that was not taken into account during off-line compilations.

Our approach differs from the formalization of correctness of complex plans in [LLL$^+$94] in that we consider exogenous actions and our control modules are reactive. Our work can be seen as strengthening the results in [Dru89, Sch87] in the sense that

- Schoppers in Section 6.1 of [Sch87] says, "Universal plans not only anticipate every possible situations in a domain but actually prescribe an action for every initial state"; more over the prescribed action is *usually optimal. In our formulation the prescribed action is optimal.*
- Drummond in [Dru89] says, "sound SCRs guarantee that local execution choices always lead to *possible goal achievement*". *In our formulation, local execution choices always lead to goal achievement.*

Finally, we deflect criticisms of universal plans by having the control rules only with respect to a set of highly plausible states, not the set of all states. The later is often prohibitively large and unmanageable. Finally, we consider explicit sensing actions, and conditional plans, which to the best of our knowledge is not considered in any research on situation control rules or universal plans. It is considered in the language $\mathcal{R}$ in [Lev96], but they do not allow exogenous actions, and plans in $\mathcal{R}$ are not reactive and are not appropriate for dynamic domains where change may happen without the agents own actions.

In the previous paragraphs we talked about an agent being controlled by continuously executing a control module. The actions in the right hand side of control rule in a control module can itself be implemented by another control module, and so on.

In case of a mobile robot in an office environment (and most applications), there could be several hierarchies of such control modules. Although, our theory is appropriate for the higher level control module, because of high uncertainty in the lower levels of sensing and action in a mobile robot[5], it is not appropriate for the lower levels, such as, for executing an action to go straight until an intersecting corridor is detected.

[5]This may not be the case for many other agents, such as an agent in the Internet environment. In that case our formulation will be applicable at all levels.

For the lowest level, we did use control rules, but not exactly as mentioned in the earlier paragraphs. We used the rules and the formalism described in [SKR95] that has multi-valued logic and is geared to handle uncertainties. In particular, the formulation of blending of control modules [SKR95] was extremely useful in our robot, in providing smooth motion while blending the actions of following a corridor and avoiding an obstacle.

3 Top-down design of our robot

In this section we first describe our representation of the environment. We then describe the different levels of the robot control and relate the role of agent theories in that level. Finally, we put everything together and describe the overall architecture of the robot.

3.1 Representing the environment

We partitioned the topological map into several regions each labeled by a landmark name based on two criteria - (i) so that the landmarks can be recognized by the robot through sensing and memory about its past positions, and (ii) there were enough landmarks that the robot could achieve the required task and recover from any failures.

Based on the above criteria, we decided to designate areas of (approximately) constant corridor width within the navigation domain as landmarks. When a robot moves from one landmark to another, the corridor width changes abruptly (see Figure 2). If the landmark is a break in the wall, like a junction or door, it is designated as a landmark of type *junction*. If the landmark is a detected corridor, that is, it detects it is navigating within a hallway of a given width, it is designated as a landmark of type *corridor*. Entrances to rooms are designated as landmarks of type *door*.

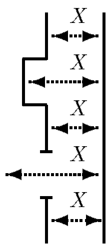

Figure 2: Areas marked X are landmarks.

We then stored adjacency information about landmarks and approximate distance between adjacent landmarks in a table. We further discuss this in a later section.

3.2 The top-level module and its correctness

Our top-level module was a control module of the kind described in the previous section. Its reactive structure was geared towards gracefully recovering from breakdowns which can be modeled as exogenous actions. Following was our top-level control module:

if $\neg visit\_conf\_1$ **then** $go\_to\_conf(1)$
if $at\_conf(1), u(avail(1))$ **then** $sense\_avail(1)$
if $at\_conf(1), \neg avail(1)$ **then** $go\_to\_conf(2)$
if $at\_conf(1), avail(1), \neg visit\_prof(1)$ **then** $go\_to\_prof(1)$
if $at\_conf(2), u(avail(2))$ **then** $sense\_avail(2)$
if $at\_conf(2), avail(2), \neg visit\_prof(1)$ **then** $go\_to\_prof(1)$
if $at\_conf(2), \neg avail(2), \neg visit\_prof(1)$ **then** $go\_to\_prof(1)$
if $at\_prof(1), \neg visit\_prof(2)$ **then** $go\_to\_prof(2)$
if $at\_prof(2), \neg back\_to\_director$ **then** $go\_to\_director$
if $back\_to\_director$ **then** $HALT$

In the above control module the part between **if** and **then** is a list of fluents, whose value is determined by sensing. The function u is used to denote that the value of a particular fluent is unknown. We would like to reiterate that, to the best of our knowledge, none of the previous work on control rules or universal plans allow such a construct. Even in [SKR95], where multi-valued logic is used, the truth value of a fluent ranges from 0 to 1, and does not take into account that the fluents truth value may be unknown. Also, special sensing actions such as the action $sense\_avail(1)$ in the above module is not used in earlier work on control modules or universal plans. They have been used in the reasoning about actions [SL93, Lev96] and planning community [GW96] though, but there also not in any reactive framework. *We must say that we appreciate the organizers of the contest who foresaw this scientific advance when creating the contest problem.*

Note that the above control module could be viewed as an universal plan for the subplan-sequencer but it is by no means an universal plan for the whole event. This module alone is not sufficient for the robot to recover from failures, e.g. when it gets stuck and needs to be restarted. It was enough, however, because we can input the position and the heading of the robot through the keyboard when we need to restart.

We constructed the above control module manually. *But we were able to use our agent theory described in the previous section to verify the correctness of our control module.* Note that the goal of this control module can not be simply expressed as a collection of fluents (not even as a classical formula of fluents), as is normally required in most planning systems. It can only be expressed as a formula with temporal and knowledge operators. Here also, although goals with temporal operators have been considered in the agents community [Sin94], in the planning community it was only recently studied in [BK96]. But, the incorporation of both temporal and knowledge operators in a goal has never been formally studied, except in a preliminary attempt in [GW96].

We now give a declarative representation of our goal in an extension of the language FMITL (First-order metric interval temporal logic) [BK96]. Our extension allows specification of knowledge. The meaning of various operators and atoms in the following specifications are: Ka means a is known to be true; $avail(1)$ means conference room 1 is available; $infprof1(1)$ means professor 1 has been informed that the meeting will be in conference room 1; $\Box f$ means always f is true; $\Diamond f$ means eventually f is true; and $at(dir)$ means the robot is at the directors office.

$$
\begin{aligned}
&[Kavail(1) \vee K\neg avail(1)] \wedge [K\neg avail(1) \Rightarrow (Kavail(2) \vee K\neg avail(2))] \wedge \\
&[Kavail(1) \Rightarrow (infprof1(1) \wedge infprof2(1) \wedge infdirector(1))] \wedge \\
&[(Kavail(2) \wedge K\neg avail(1)) \Rightarrow (infprof1(2) \wedge infprof2(2) \wedge infdirector(2))] \wedge \\
&[(K\neg avail(1) \wedge K\neg avail(2)) \Rightarrow (infprof1(dir) \wedge infprof2(dir) \wedge infdirector(dir))] \wedge \\
&[\forall X\, avail(X) \Rightarrow \Box avail(X)] \wedge [\Box clear\_from\_obstacle \wedge \Diamond\Box at(dir)]
\end{aligned}
$$

We used our agent theory to verify the correctness of goals which are formulas with temporal and knowledge operators, by checking if the unfoldings of the control modules with respect to the various states, satisfied the goal. But, due to temporal operators in the goal the trajectory of the states became as important as the the final state and because of that our automatic construction algorithm was not able to construct control modules for such goals. *One of our future goals is to find algorithms to automatically generate control modules for complex goals with knowledge and temporal operators.*

3.3 Modules to take from one room to another room - automatic construction

The top level control module used two special sensing actions: $sense\_avail(1)$ and $sense\_avail(2)$, and five non-sensing actions: $go\_to\_conf(1)$, $go\_to\_conf(2)$, $go\_to\_prof(1)$, $go\_to\_prof(2)$, and $go\_to\_director$. The two sensing actions were implemented using sonars that checked if there was any substantial movement in the room, and did not have a reactive structure. The non-sensing actions were abstracted as going from one landmark (or node) in the topological map to another. This was implemented using a control module. We now describe this control module, which we refer to as the navigation control module.

The navigation control module was basically a table with three columns: current node[6], goal node, and next adjacent node. An entry (5,3,4) in that table meant that if the robot is currently at node 5 and its ultimate goal is to reach node 3 then it should next go to the adjacent node 4. The actions in this module were actions that took the robot from any node to its specified adjacent node. *This control module was automatically generated and we used the approximate distance between adjacent nodes as the cost of the action that takes the robot from one of those nodes to the other.*

The actions that took the robot from one node to one of its specified adjacent node was also implemented by a control module. We now describe this control module.

3.4 Control module for going to an adjacent node - correctness

The actions used in this control module were of two types: turning actions, that change the heading of the robot and navigational actions, that take the robot from one kind of node to an adjacent node. The turning actions are listed in the third column of Table 2 and the navigation actions are listed in the third and sixth columns of Table 1.

Following gives a flavor of how the effect of navigation and turning actions could be formally specified:

$detect\_corridor$ **causes** $at(X), \neg at(Y)$ **if** $at(Y), type(Y, junction), type(X, corridor), heading(dir), adj(X, Y, dir)$

$turn\_180$ **causes** $heading(west), \neg heading(east)$ **if** $heading(east)$

In the above specification, $type(X, corridor)$ means that node X is of the type corridor. Similarly, $adj(X, Y, dir)$, means that the nodes X and Y are adjacent and to go from node X to node Y the robot's heading must be dir. We now give a simple map with five nodes and give a table that has the adjacency and heading information for the given map. Intuitively, the first row of Table 3 can be read as node 5 is of type corridor, node 4 is of type junction, node 5 and 4 are adjacent, approximate distance between them is 150 units, and to go from node 5 to node 4 the robot's heading must be west.

Now let us get back to the control module that takes the robot from one node to a specified adjacent node. The condition part of each control rule in the control module checks if the robots heading

[6]We use the terms 'landmark' and 'node' interchangeably.

Type of **From Node**	Type of **To Node**	Action
corridor	junction	*detect_break*
junction	corridor	*detect_corridor*
X	Y, Y=X	*go_forward*
junction	door	*go_to_door*
door	junction	*go_to_door*

Table 1: Some action rules for generation of navigation actions

Current Heading	Need to be	Action	Current Heading	Need to be	Action
east	west	*turn_180*	north	west	*turn_left_90*
east	south	*turn_right_90*	north	east	*turn_right_90*
east	north	*turn_left_90*	north	south	*turn_180*
west	east	*turn_180*	south	east	*turn_left_90*
west	south	*turn_left_90*	south	west	*turn_right_90*
west	north	*turn_right_90*	south	north	*turn_180*
X	Y, Y=X				

Table 2: Action rules for generation of turning actions

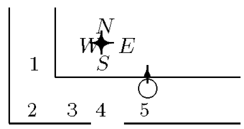

Figure 3: A sample map.

is appropriate (as specified in Table 3), and if not, the robot executes the turning action specified by the Table 2. If the heading is appropriate the robot executes the navigation action specified by Table 1. In other words, Tables (1 and 2) are compiled from the description of actions and describe which action to take given particular information about an initial state and a goal. *Tables (1 and 2) were generated manually and their correctness was verified using the theory of the paper.* We could have, however, automatically constructed them using our algorithm.

3.5 Basic actions and Skills

We refer the actions in the previous control module (the actions in the third column of the tables 1 and 2) as the basic actions. They were implemented by *merging* several skills which were implemented using *control modules based on multi-valued logic.* At the lowest level, the uncertainty

N1	Type N1	N2	Type N2	Heading	Dst.
5	corridor	4	junction	west	150
4	junction	3	corridor	west	100
3	corridor	2	wall_break	west	150
2	wall_break	1	corridor	north	150

Table 3: An example showing adjacency records for the domain depicted by Figure 3.4.

associated with the physical world makes it harder to implement low level tasks using control modules of the kind (where fluents are 2-valued) described in this paper. We found the control modules of the kind described in [SKR95][7] that uses multi-valued logic to be more appropriate for that purpose. We also used the concept of 'merging' control described in [SKR95] for blending skills into basic actions. For example, the basic action of going forward means translating forward with a certain speed and at the same time avoiding obstacles, and avoiding a crash. I.e., the basic action go_forward involves the *blending* (or *normalization*) [SKR95] of many of the skills (not necessarily all) with respect to assumed associated skill priorities: *Avoid Crash, Control Steering, Control Speed, Facing Ahead, Avoid Obstacles, Search for Destination.* The tables (4 and 5) list the basic actions of our robot, the skills of our robots and their intuitive function. Table 5 also lists the weight of the various skills for implementing the action *detect_corridor*.

Action Name	Effect
go_forward	Go forward for a given distance.
detect_corridor	Go forward until a corridor of a given width is detected.
go_to_door	Go forward some distance to enter a room.
detect_break	Follow a corridor of a given width until a change in the corridor width is detected.
turn_left_90	Turn left 90 degrees.
turn_right_90	Turn right 90 degrees.
turn_180	Turn 180 degrees.
speak	Transmit information.
detect_occupied	Sensing action to detect if a room is occupied.
align	Correct hardware uncertainty.

Table 4: Basic actions

Behavior β_i	Function	Weight
Translate_Forward	Always traverse forward.	10
Point_To_Opening	Find openings for the robot to traverse through.	15
Follow_Corridor	Follow a virtual corridor and look for a hallway of a given width.	15
Avoid_Close_Sonar	Avoid close sonar sensor readings.	20
Avoid_Close_InfraRed	Avoid close infra red sensor readings.	110
Stop_When_Touched	Stop when bumped.	1000
Avoid_Crashing	Stop if in danger of crashing into obstacles.	1000
Point_Forward	Keep robot from turning back.	1000

Table 5: Some of the skills of our robot and weights for *detect_corridor*

The concept of normalization of skills to define a basic action can be formally described as follows: Suppose, $\mathcal{A}$ is the set of basic actions and $\mathcal{B}$ is the set of skills, and for each $\beta \in \mathcal{B}$ there is an

[7]Although, Saffioti et al [SKR95] have a notion of correctness for their control modules, they do not consider exogenous actions, and sensing actions. *One of our future goal is to extend our approach to the control modules in [SKR95] in defining correctness in presence of exogenous and sensing actions, as well as develop methods to automatically construct such control modules.*

associated translation value t_β and rotation value r_β.

Then for each basic action $a \in \mathcal{A}$ which is the normalization of behaviors $\beta_1, \ldots, \beta_n$ we associate to a the weights $w_1^a, \ldots, w_n^a$ and compute the translation and rotation by the following formula:

$$t_a = \frac{\sum_{i=1}^n t_{\beta_i} \cdot w_i^a}{\sum_{i=1}^n w_i^a} \qquad \text{and} \qquad r_a = \frac{\sum_{i=1}^n r_{\beta_i} \cdot w_i^a}{\sum_{i=1}^n w_i^a} \tag{1}$$

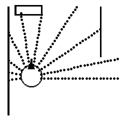

Figure 4: A diagram showing sonar emissions when avoiding an obstacle.

The following example illustrates one such normalization.

Example 1 Suppose our robot wants to perform the action *detect_corridor* in the situation depicted by Figure 4 with current velocity of 150 mm/second and heading index of 0 (which corresponds to the robot heading directly North). The action *detect_corridor* is composed of the skills listed in Table 5.

Suppose the current sensor readings cause the corresponding behaviors to propose individual translations and rotations as follows:

Translate_Forward proposes $t_{\beta_1} = 150; r_{\beta_1} = 0;$
Point_To_Opening proposes $t_{\beta_2} = 90; r_{\beta_2} = 150;$
Follow_Corridor proposes $t_{\beta_3} = 0; r_{\beta_3} = 0;$
Avoid_Close_Sonar proposes $t_{\beta_4} = -100; r_{\beta_4} = 65;$
Avoid_Close_InfraRed proposes $t_{\beta_5} = -100; r_{\beta_5} = 65;$
Stop_When_Touched proposes $t_{\beta_6} = 0; r_{\beta_6} = 0;$
Avoid_Crashing proposes $t_{\beta_7} = 0; r_{\beta_7} = 50;$
Point_Forward proposes $t_{\beta_8} = 150; r_{\beta_8} = 0;$

The overall translation and rotation for the robot would be $t_{detect\_corridor}$=44.1 and $r_{detect\_corridor}$=3.3. Therefore the action *detect_corridor*, through its corresponding behaviors, causes the robot to slow down to 44 mm/sec and turn right 3.3 units (which corresponds to about 1 degree) to avoid the encountered obstacle.

3.6 The overall architecture

Figure 5 depicts the overall architecture of Diablo which is very much like the three tired architecture in [BKMS95]. We now relate this architecture with the modules described in the previous sections and show the correspondence between them.

In Figure 5 the Task-list block corresponds to the top level module. The actions of that module are referred to as tasks in our architecture and also in [BKMS95]. The various tasks that we have are: $sense\_avail(1)$, $sense\_avail(2)$, $go\_to\_conf(1)$, $go\_to\_conf(2)$, $go\_to\_prof(1)$, $go\_to\_prof(2)$, and $go\_to\_director$. Execution of the last five tasks involved the *navigation control module* which would take the robot from one node to another. This corresponds to the shortest path generator

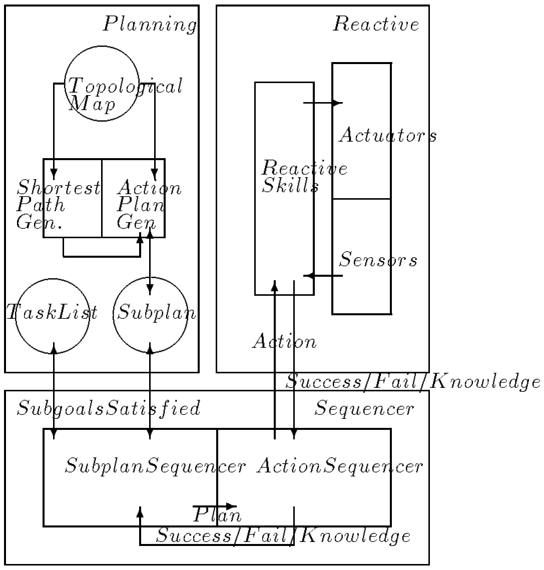

Figure 5: A depiction of UTEP3AI, the architecture used for the AAAI '96 Robot Competition.

block in the architecture. The control module that takes the robot from one node to an adjacent node corresponds to the action plan generator block in our architecture. The universal plan block in the architecture corresponds to the combination of the navigation control module and the control module that takes the robot from one node to its adjacent node. The Sequencer as a whole would take the appropriate task from the task list and use the universal plan to determine the sequence of basic actions to be executed. The reactive block in the architecture corresponds to the basic action and skills module.

The above correspondence gives an idea of the process behind going from a theoretical hierarchy of control modules to a control architecture. It suggests that merging of levels of control modules may be useful and necessary from the implementation point of view. In fact to maintain reactivity, we need to merge levels of control modules.

4 Diablo in AAAI 97 Competition

In this section, we show how Diablo's architecture can be easily modified and extended in such a way that Diablo can complete new tasks. We believe that this is one of the main advantages of Diablo. It may be worth noting that the modification for the new competition took only two weeks and it was done by two students; one worked with the new actions and the other with the generation of plans[8]. Let us start with a brief description about the contest.

In the home vacuum contest, the domain consists of a corridor and three rooms: a living room, a family room, and a bedroom. The robot can travel along the corridor to enter the rooms. In the "Tidy Up" event the robot starts at the disposal station. Humans will periodically enter and leave the room (possibly leaving messes). It is the robot's job to keep the room clean. Robot must not enter a room when a human is present (but humans will not enter a room while the robot is cleaning). During a given 15 minute run, humans will be present in each room for 1 minute out of

[8]Our robot was down for some unknown reason and could only be used two weeks before the competition started.

every 5, except for the living room which will be occupied 3 minutes out of every 5. Robots may clean messes in the hallway while humans are present there.

In order to complete the new task autonomously, Diablo needs two additional capabilities:

- Detecting the mess;
- Navigation inside the room.

Therefore, our first task is to add new actions into the set of actions of Diablo. The first action is for detecting the mess and the second action allows Diablo get as close to a wall as possible. The second action is used when the robot navigates inside a room. The planner and plan sequencer do not require any changes. The top level control module for the event is as follows.

if $at(disposal)$ **then** $go\_to(1)$
if $at(1), u(avail(1))$ **then** $sense\_avail(1)$
if $at(1), avail(1), u(clean(1))$ **then** $clean(1)$
if $at(1), \neg avail(1)), u(clean(2))$ **then** $go\_to(2)$
if $at(1), \neg avail(1)), \neg clean(2)$ **then** $go\_to(2)$
if $at(1), \neg avail(1)), clean(2), u(clean(3))$ **then** $go\_to(3)$
if $at(1), \neg avail(1)), clean(2), clean(3)$ **then** $SUSPEND$
......

Due to the limited space we do not list here the complete control module for the event (it contains the control rules for the robot at room 2 and 3) but it is easy to verify that the module indeed ensures that the robot continuously travels between rooms and only stops when all three rooms have been cleaned.

5 Conclusions

We have presented the architecture of our robot which has been used successfully in two robot competitions. We believe that the success is due to its simple architecture, its flexibility, and the correctness of the top level control module. The main disadvantage of Diablo is that it does not have the capability to recognize the environment from what it can see, say by using the sensors or the camera. Thus, it cannot work in unknown environment. To add this feature to Diablo is one of our main goals in the near future. By doing that we hope that Diablo will be able to complete more challenging tasks which require more than navigation and simple sensing capabilities. Another goal is the implementation of an algorithm that automatically generates the top level control modules which have been done manually until now.

References

[Ark91] R. Arkin. Integrating behavioral, perceptual, and world knowledge in reactive navigation. In P. Maes, editor, *Designing Autonomous Agents*, pages 105–122. MIT Press, 1991.

[BGP96] C. Baral, A. Gabaldon, and A. Provetti. Formalizing Narratives using nested circumscription. In *AAAI 96*, 1996.

[BK96] F. Bacchus and F. Kabanza. Planning for temporally extended goals. In *AAAI 96*, pages 1215–1222, 1996.

[BKMS95] R. Bonasso, D. Kortencamp, D. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. In M. Woolridge, J. Muller, and M. Tambe, editors, *Intelligent Agents - II*, pages 187–202. Springer, 1995.

[BS96] C. Baral and T. Son. Relating theories of actions and reactive robot control. In *Proc. of AAAI 96 Workshop on Reasoning about actions, planning and robot control: bridging the gap.*, 1996.

[BS97a] C. Baral and T. Son. Regular and special sensing in robot control - relation with action theories. In *Proc. of AAAI 97 Workshop on Robots, Softbots, and immobots - Theories of Action, Planning and Control (to appear)*, 1997.

[BS97b] C. Baral and T. Son. Relating theories of actions and reactive control. Technical Report, CS Dept., University of Texas at El Paso (http://cs.utep.edu/chitta/chitta.html), 1997.

[Dru89] M. Drummond. Situation control rules. In *KR 89*, pages 103–113, 1989.

[GL93] M. Gelfond and V. Lifschitz. Representing actions and change by logic programs. *Journal of Logic Programming*, 17(2,3,4):301–323, 1993.

[GW96] K. Golden and D. Weld. Representing sensing actions: the middle ground revisited. In *KR 96*, pages 174–185, 1996.

[KL94] G. Kartha and V. Lifschitz. Actions with indirect effects: Preliminary report. In *KR 94*, pages 341–350, 1994.

[KNH97] D. Kortenkamp, I. Nourbaksh, and D. Hinkle. The 1996 aaai mobile robot competition and exhibition. *AI magazine*, pages 25–32, 1997.

[Lev96] H. Levesque. What is planning in the presence of sensing? In *AAAI 96*, 1996.

[LLL+94] Y. Lesperance, H. Levesque, F. Lin, D. Marcu, R. Reiter, and R. Scherl. A logical approach to high level robot programming – a progress report. In *Working notes of the 1994 AAAI fall symposium on Control of the Physical World by Intelligent Systems (to appear), New Orleans, LA*, November 1994.

[Mae91] P. Maes, editor. *Designing Autonomous Agents*. MIT/Elsevier, 1991.

[San94] E. Sandewall. The range of applicability of some non-monotonic logics for strict inertia. *Journal of Logic and Computation*, 4(5):581–616, october 1994.

[Sch87] M. Schoppers. Universal plans for reactive robots in unpredictable environments. In *IJCAI 87*, pages 1039–1046, 1987.

[Sin94] M. Singh. *Multiagent systems - a theoretical framework for intentions, know-how, and communications*. Springer-Verlag, 1994.

[SKR95] A. Saffioti, K. Konolige, and E. Ruspini. A multivalued logic approach to integrating planning and control. *Artificial Intelligence*, 1995.

[SL93] R. Scherl and H. Levesque. The frame problem and knowledge producing actions. In *AAAI 93*, pages 689–695, 1993.