ViTAPlan: A Visual Tool for Adaptive Planning

Dimitris Vrakas and Ioannis Vlahavas

Department of Informatics, Aristotle University
54124 Thessaloniki, Greece
Email: {dvrakas, vlahavas}@csd.auth.gr

Abstract. This paper presents a friendly visual tool for HAP, a rule-configurable planning system, which automatically adapts to each problem, in order to achieve best performance. HAP analyzes the problem and uses a rule system in order to configure the planning parameters in a way that best suites the morphology of the problem. The visual tool enables the user to use the planning system, get advice from the built-in rule system and even interfere with it. ViTAPlan also contains a visual designer, based on the Planning Domain Definition Language, that enables the user to create new planning domains and problems in a graphical way and get visual representations of existing ones. Furthermore the tool contains a module that simulates the execution of the plan and illustrates the changes in the world, which follow the application of each action in the plan.

1 Introduction

Automated Planning has been an active research topic for almost 40 years and during these four decades a great number of papers describing new methods, techniques and systems have been presented that mainly focus on ways to improve the efficiency of planning systems. However, there are not many successful examples of planning systems adapting to industrial use. From a technical point of view, this can be mainly explained by two facts: a) the planning systems are not yet efficient enough to handle real-world problems and b) since the end-user of a planning system in the industry will not be a planning-expert, systems must be accompanied by user friendly interfaces.

Concerning the efficiency of planning systems, the major part of researchers focus on domain–independent planning systems trying to make them as efficient as possible, concerning both planning time and length of produced plans. Although, there have been examples of really efficient systems, during the last decade, there are still open issues to be addressed. Little systems support aspects of planning that are crucial to industry, such as temporal planning or efficient handling of resources. For instance, Advisor (Marinagi et al 1996) is a successful case of applying a planning system in real world applications. The planning system embodies a symbolic constraint solver and a temporal reasoning mechanism in order to allow the expressiveness needed for encoding the problems. Another obstacle in the application of planning systems is the fact that they exhibit instabilities in their efficiency among different domains or even

problems of the same domain. A planner may be very good in specific domains and problems but there is no planning system that guaranties a general top performance.

As far as user interfaces are concerned, there have been several approaches from institutes and researchers to create visual tools for defining problems and running planning systems, such as the GIPO system[1], the SIPE-2[2] and the ASPEN[3] graphical user interfaces. Moreover, there is a number of approaches in building visual interfaces for specific applications of planning. The PacoPlan project[4] aims in building a web-based planning interface for specific domains. AsbruView (Kosara and Miksch, 2001) is a visual user interface for time-oriented skeletal plans representing complex medical procedures. Another example of visual interfaces for planning is the work of the MAPLE research group at the university of Maryland (Kundu et al, 2002), which concerns the implementations of a 3D graphical interface for representing hierarchical plans with many levels of abstractions and interactions among the parts of the plan. Although these approaches are very interesting and provide the community with useful tools for planning, there is still a lot of work to be done in order to create an integrated system that meets the needs of the potential user.

This paper describes ViTAPlan, a visual tool for a the HAP (Highly Adjustable Planner) system, which enables the user to setup it by tuning several planning parameters. The system is also equipped with a rule system able to automatically fine-tune the planner based on the morphology of the problem in hand. The graphical interface enables the user to setup and run HAP, get advice from the rule system and also design new domains and problems. Finally the tool enables the user to view visual representations of the plan and preview a simulation of the execution of it in the problem's world. The graphical interface is a first prototype of a project aiming in producing an integrated planning system for use in real world situations.

2 Using HAP

HAP, is a highly adjustable planning system that can be customized by the user through a number of parameters. These parameters concern the type of search, the quality of the heuristic and several other features that affect the planning process. The HAP system is based on the BP (Bi-directional Planner) planning system (Vrakas and Vlahavas, 2001) and uses an extended version of the ACE (ACtion Evaluation) heuristic (Vrakas and Vlahavas, 2002).

HAP is capable of planning in both directions (progression and regression). The system is quite symmetric and for each critical part of the planner, e.g. calculation of mutexes, discovery of goal orderings, computation of the heuristic, search strategies etc., there are implementations for both directions. The *direction* of search is the first adjustable parameter of HAP used in tests, with the following values: a) 0 (Regression or Backward chaining) and b) 1 (Progression or Forward chaining).

[1] http://scom.hud.ac.uk/planform/gipo/
[2] http://www.ai.sri.com/~sipe/gui.html
[3] http://www-aig.jpl.nasa.gov/public/planning/aspen/aspen_index.html
[4] http://lpis.csd.auth.gr/projects/pacoplan/

As for the search itself, HAP adopts a weighted A* strategy with two independent weights: w_1 for the estimated cost for reaching the final state and w_2 for the accumulated cost of reaching the current state from the starting state (initial or goals depending on the selected direction). For the tests with HAP, we used four different assignments for the variable *weights* which correspond to different assignments for w_1 and w_2: a) 0 (w_1=1, w_2=0), b) 1 (w_1=3, w_2=1), c) 2 (w_1=2, w_2=1) and d) 3 (w_1=1, w_2=1).

The size of the planning agenda (denoted as *sof_agenda*) of HAP also affects the search strategy and it can also be set by the user. For example, if we set *sof_agenda* to 1 and w_2 to 0, the search algorithm becomes pure Hill-Climbing, while by setting *sof_agenda* to 1, w_1 to 1 and w_2 to 1 the search algorithm becomes A*. Generally, by increasing the size of the agenda we reduce the risk of not finding a solution, even if at least one exists, while by reducing the size of the agenda the search algorithm becomes faster and we ensure that the planner will not run out of memory. For the tests we used three different settings for the size of the agenda: a) 1, b) 100 and c) 1000

The *OB* and *OB-R* functions introduced in BP and ACE respectively, are also adopted by HAP in order to search the states of the search for violations of orderings between the facts of either the initial state or the goals, depending on the direction of the search. For each violation contained in a state, the estimated value of this state that is returned by the heuristic function, is increased by violation penalty, which is a constant number supplied by the user. For the experiments of this work we tested the HAP system with three different values of *violation_penalty*: a) 0, b) 10 and c) 100.

The HAP system employs the heuristic function of the ACE planner, plus two variations of it, which are in general more fine-grained. There are implementations of the heuristic functions for both planning directions. All the heuristic functions are constructed in a pre-planning phase by performing a relaxed search in the opposite direction of the one used in the search phase. During this relaxed search the heuristic function computes estimations for the distances of all grounded actions of the problem.

The user may select the heuristic function by configuring the *heuristic_order* parameter. The three acceptable values are: a) 1 for the initial heuristic, b) 2 for the first variation and c) 3 for the second variation.

HAP also embodies a technique for simplifying the definition of the sub-problem in hand. This technique eliminates from the definition of the sub-problem (current state and goals) all the goals that have already been achieved in the current state and do not interfere in any way with the achievement of the remaining goals. In order to do this the techniques performs, off-line before the search process, a dependency analysis on the goals of the problem. The parameter *remove_subgoals* is used to turn on (value 1) and off (value 0) this feature of the planning system.

The last parameter of HAP is *equal_estimation*, which defines the way in which states with the same estimated distances are treated. If *equal_estimation* is set to 0 then between two states with the same value in the heuristic function, the one with the largest distance from the starting state (number of actions applied so far) is preferred. If *equal_estimation* is set to 1, then the search strategy will prefer the state, which is closer to the starting state.

The proposed graphical interface enables the user to use HAP with a friendlier and more accurate way. From the initial screen of the interface, which is shown in Figure 1, the user uses common dialogues in order to browse for the domain and problem

files. The plan is also presented in the same screen along with statistics concerning the planning process (planning time, length of solution, examined states). Optionally, the user may select to configure HAP, through the window shown in Figure 2, by selecting through a number of options for each planning parameter.

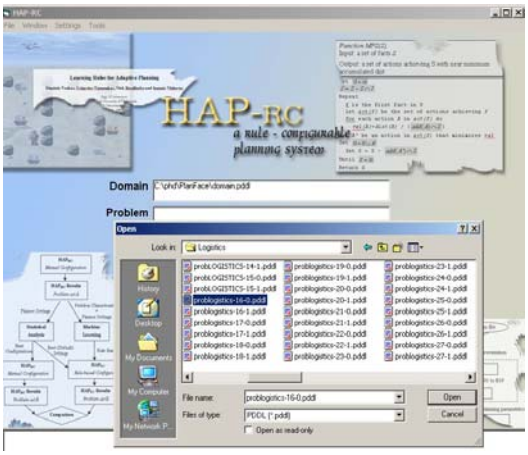

Figure 1. Selecting Domain and Problem

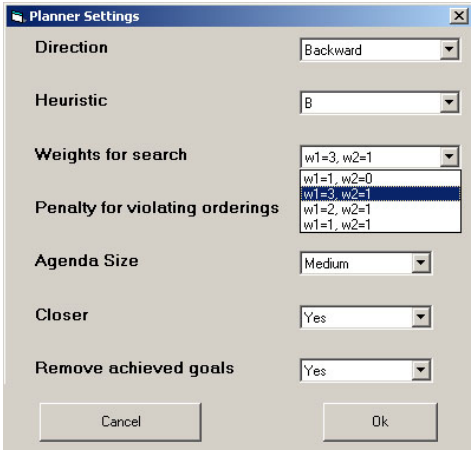

Figure 2. Configuring HAP

3 Automatic Configuration of HAP

HAP-RC (Vrakas et., al 2003) is an extension to the HAP planning system, which uses a rule system in order to automatically select the best settings for each planning parameter, based on the morphology of the problem in hand. HAP-RC, whose architecture is outlined in Figure 3 is actually HAP with two additional modules (Problem Analyzer and Rule System) which are utilized off-line, just after reading the representation of the problem in order to fine tune the planning parameters of HAP.

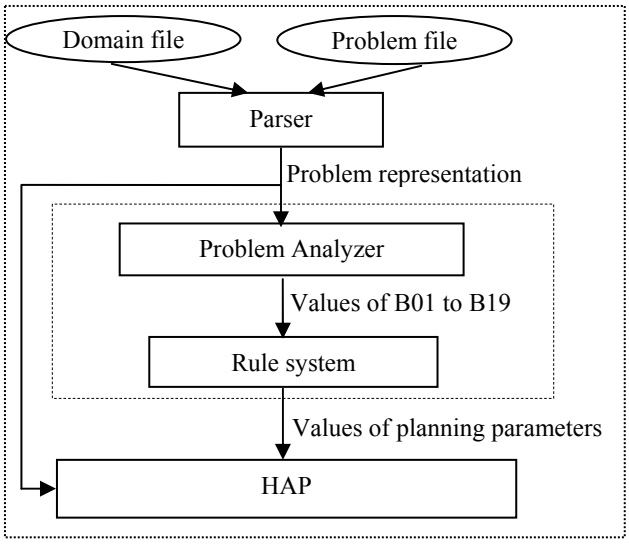

Figure 3. HAP-RC Architecture

The role of the Problem Analyzer is to identify the values of a specific set of 19 problem characteristics (noted as B1 to B19). These characteristics include measurable attributes of planning problems, such as number of facts per predicate or branching factor of the problem e.t.c. After the identification of the values of the attributes, which may requires a limited search in the problem, the analyzer discretizes the results in three categories (small, medium and large) and feeds the Rule system with a vector containing the discretized values for the 19 problem attributes.

The Rule system contains a number of rules that combine specific values of the problem attributes with settings of the planning parameter that result in better planning performance (shorter plans in less planning time). These rules have been extracted from Machine Learning techniques on data produced by thorough experiments with the HAP system. More specifically, we tested all the possible combinations of the parameters of HAP on a set of 150 problems and for each run we kept record of the values of the problem attributes, the specific setup for HAP and the value for a metric combining planning time and plan length. The data set was then fed to a Machine Learning tool in order to learn a rule-based classification model that would discriminate between good and bad value of the metric based on the rest of the attributes.

ViTAPlan also provides the user with the option to use the Problem Analyzer and the Rule System of HAP-RC in order to automatically fine-tune the planning parameters of HAP. The relevant window of the interface is shown in Figure 4. This window is divided in three parts: a) the first part shows the discretized values for the 19 problem characteristics, as produced by the Problem analyzer, along with a description for each one through hot spots. b) the second part provides the user with the list of the triggered rules, i.e. the rules which refer to values for B1 to B19 that comply with the values produced by the Problem Analyzer. c) the last part provides the user with the proposed values for the planning parameters of HAP.

Apart from the obvious usage of the specific part of the interface, which is to automatically fine-tune HAP, it can also help an advanced user (knowledge engineer) to better understand the morphology of the problem by looking at the values of the problem attributes. It also enables the advanced user to alter the results of applying the rules, by manually selecting the subset of the triggered rules that will be eventually fired. The rule system embodies a conflict resolution strategy, which is based on the confidence and the support of each rule. The interface enables the user to see the subset initially selected for firing through check boxes at the left of each rule. The user can deselect and select rules, while the interface automatically deactivates all the other rules that are in conflict with the selected ones.

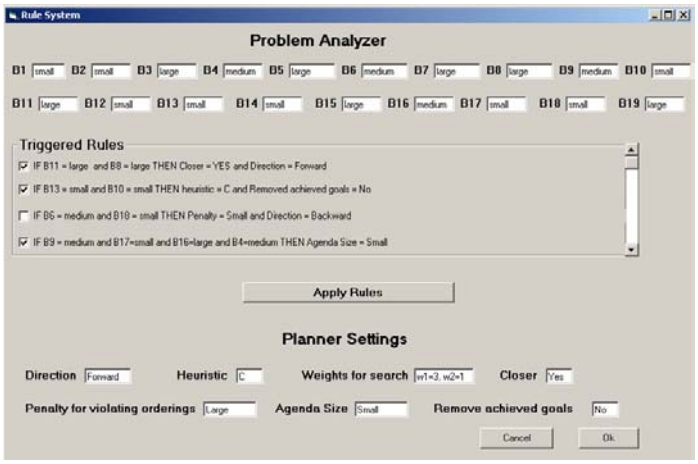

Figure 4. Using the Rule System

Consider for instance, the example of Figure 5, where there are four triggered rules, since {B4, B6, B8, B9, B10, B11, B13, B16, B17, B18} = {medium, medium, large, medium, small, large, small, large, small, small} in the problem being analyzed. However, rules 1 and 3 are in conflict, since the first rule proposes value *Forward* for the *Direction* parameter, while the third one proposes a different value (*Backward*) for the same parameter.

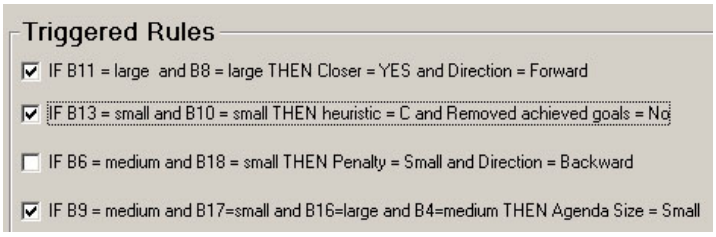

Figure 5. Initial firing subset

The rules in HAP-RC have been sorted in decreasing order of confidence and support and the conflict resolution strategy works in a greedy way, selecting for firing the

first rule from the top of the list that is not in conflict with the rules already selected for firing. Therefore, the rules proposed for firing are indicated with ticks in the corresponding check boxes. By clicking in the third rule the interface automatically includes it in the firing subset and removes the rules that are in conflict with the selection of the user resulting in the selection shown in Figure 6.

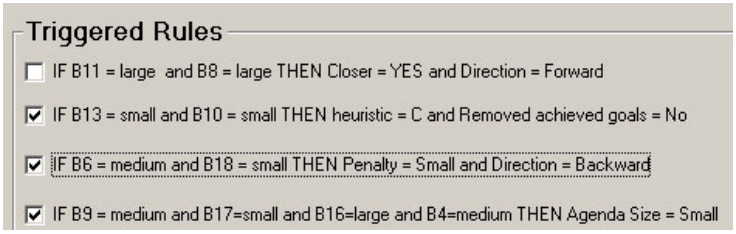

Figure 6. Final firing subset

4 Designing Domains and Problems

The proposed visual tool enables the user to view and design new domains and problems through a visual representation. In order to build a new domain or problem the user can add new structural elements, like object classes, predicates or operators, and make all the necessary assignments with simple movements of the mouse. The interface is responsible for checking the validity of the user's design and generating the appropriate PDDL (Ghallab et al 1998) files.

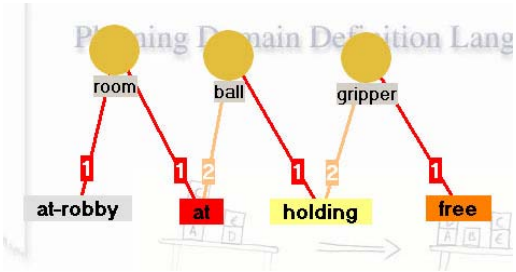

Figure 7. The predicates design for the Gipper domain

The first step for creating a new domain, is to create a design containing the objects, the predicates and the connections between them. Figure 7 illustrates this design for the *Gripper* domain, which was used in the AIPS-98 planning competition. There are three object classes in the domain, namely *room ball* and *gripper*, that are represented with circles. The domain has four predicates (*at*, *at-robby*, *holding* and *free*) that are represented with rectangles. Note here that although PDDL, requires only the arity for each predicate and not the type of objects for the arguments, the interface obliges the user to connect each predicate with specific object classes and this is used for the consistency check of the domain design. According to the design of Figure 7,

the arity of predicate *holding*, for example, is two and the specific predicate can only be connected with one object of class *ball* and one object of class *gripper*.

The constraints imposed by the design of predicates and object classes are dynamically inherited in the design of operators and problems. For example if the user want to add a fact *f* in the initial state of a problem, which makes use of the *holding* predicate, *f* will have two available edges that will be able to connect with an object of class *ball* and another object of class *gripper*. However, the interface enables the user to dynamically change the design of the predicates. In such a case, all the operators and states that make use of altered predicates and object classes are automatically updated accordingly.

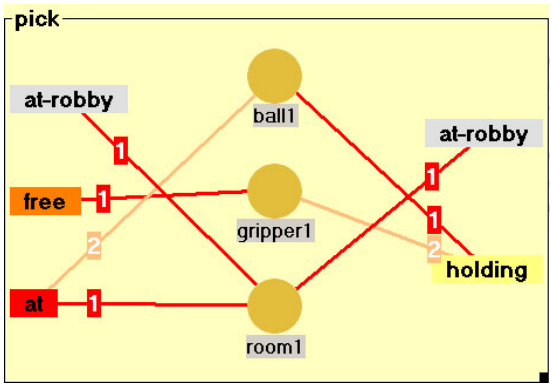

Figure 8. Operator Pick for the Gripper domain

The second step in generating a new domain is the designing of the domain's operators. Each operator, in the interface, is represented with a labeled frame, which contains a column of object classes in the middle, two columns of predicates at the two sides of it and connections between the object classes and the predicates. Figure 8, for example, illustrates the design of the *pick* operator for the *gripper* domain. The object classes in the middle column of the operator, represent the parameters of the operator, which in the case of the pick operator are three, one of each object class. In the left column the predicates, along with the connected variables, represent the preconditions of the operator, while the predicates in the right represent the effects of the operator. The interface adopts the declarative schema for designing operators, i.e. the right column of each operator represents the state of the world after the application of the operator and not the facts that will be added and deleted to it. However, the creation of the add and delete lists of the operator is straightforward, since the facts that appear in the right column but not in the left constitute the add list. Similarly, the delete list contains the facts that appear in the left column but not in the right. The choice of the declarative model versus the procedural one, was based on the fact that the first usually results in simpler designs.

The three lists of facts for the pick operator in Figure 8 are the following:
Preconditions = {at-robby(room1), free(gripper1), at(room1, ball1)}

Add-list = {holding(ball1, gripper1)}
Delete-list { free(gripper1), at(room1, ball1)}

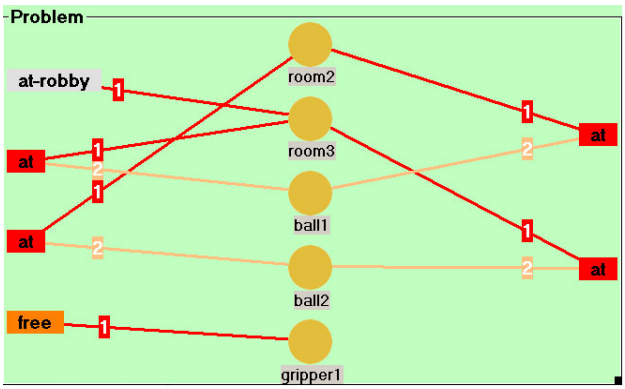

Figure 9. The design for a problem in the Gripper domain

The designing of problems in the interface follows a similar model with that of operators. Problems can be formed by creating a list of objects, two lists of predicates and a number of connections among them. Figure 9 illustrates the design of a problem in the *Gripper* domain. The objects in the middle represent the objects of the problem, the facts created by predicates of the left column represent the facts of the initial state of the problems, while the predicates in the right column represent the goals of the problem. The two states that correspond to the design of Figure 9 are the following.

Initial state = {at-robby(room3), at(room3, ball1), at(room2, ball2), free(gripper1)}
Goals = {at(room2, ball1), at(room3,ball2)}

5 Execution Simulation

One of the most important capabilities of ViTAPlan is that it visualizes the execution of the plan found by HAP. The visual tool enables the user to get visual representations of each action in the plan independently and of the whole plan. Through the first option, which is shown in Figure 10, the user may select an action from the plan and view the states of the problem's world before and after the application of the action. Therefore, the user can have a step-by-step visual representation of each intermediate state between the initial one and the goals of the problem.

The second option for the user is to view the actions of the plan on a timeline, as shown in Figure 11. The timeline for the plan presents for each action the point in which it is scheduled to be executed and the facts in its precondition and add lists. Moreover, the visualization shows the interactions between the actions in the plan. More specifically, for each action the user is able to see the preceding actions that achieved its preconditions. For instance, in the example presented in Figure 11 the third action in the plan is "drop ball1 gripper1 room2", which has two preconditions: "at-robby room2" and "holding gripper1 ball1". The first one was achieved by the second action of the plan (i.e. "move room3 room2") and the second one by the first action ("pick gripper1 ball1 room3").

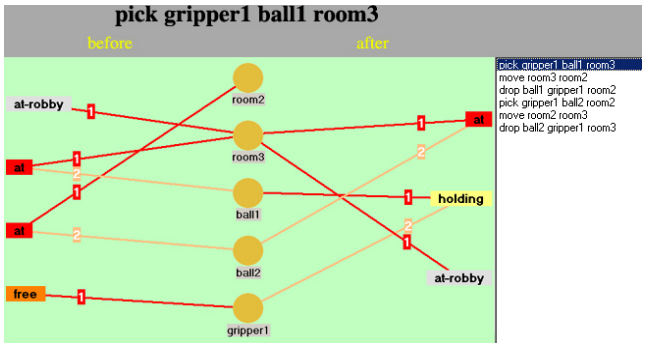

Figure 10. Visual Representation of Actions

The arcs showing the relations between the actions in the plan are very useful in order for the user to understand the complexity of the plan, find possible parallelizations and also alternatives plans. In order for the user to test if two subsequent actions can be executed in parallel he just needs to check if there are any arcs connecting these actions. Furthermore, the arcs in the graphic allow the user to understand the role of each action in the plan. In other words, he can get an idea of the objectives of each step and the reasons for the specific order in which the steps are put. For example, it is easy to see from Figure 11 that the first three actions are needed in order to have both the robot and ball1 in room2. Therefore, the user could replace this part of the plan with a possible alternative.

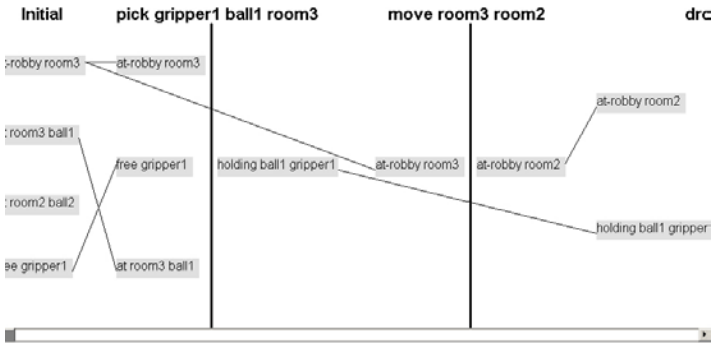

Figure 11. Plan Representation

6 Conclusions and Future Work

This paper reported on ongoing research in the field of friendly user interfaces for planning. It mainly focuses on the development of a composite interface for an adaptive planning system, which can automatically fine- tune its parameters based on rules, extracted from Machine Learning techniques, that associate planning parameters with problem attributes.

The current result of the research is ViTAPlan, a first prototype of a graphical interface for the HAP planning system, which has four main functions: a) using the

planning system through a number of windows, controls and common dialogues, which make it much easier for a non – programmer to use the planner and experiment with different setups of the planning parameters, b) use the Problem analyzer and the Rule system of HAP-RC in order to acquire useful knowledge about the morphology of each problem and automatically fine tune the planner with the most appropriate values for the planning parameters c) generating new domains and problems using a visual tool which saves the domain expert from the strict syntactic rules of PDDL, makes the definition of domains and problems more understandable, even for a non-planning-expert, and makes a number of consistency checks on the designs in order to generate PDDL files with as little flaws as possible and d) produce visual representations of the plans found by the planning system, which enable the user to better understand each step in the plan and also intervene and alter the plan at will.

In the future we plan to improve the interface in all functions of it and introduce others that will make it a complete tool for planning both for academic and industrial use. It is in our direct plans to enhance the tool for designing domains and problems with the ability to handle advanced aspects of the PDDL2.1(Fox and Long 2002), such as treatment of numerical values, explicit representation of time and duration, conditional effects e.t.c.

Acknowledgments

This research has been partially supported by SUN Microsystems, grant number: EDUD-7832-010326-GR.

References

1. Fox M., Long D. PDDL2.1 – an Extension to PDDL for Expressing Temporal Planning Domains (2002)
2. Ghallab M., et. al. PDDL – the Planning Domain Definition Language, Ver.1.3 (1998)
3. Kosara R., Miksch S. Metaphors of Movement: a Visualization and User Interface for Time-Oriented, Skeletal Plans. *Artificial Intelligence in Medicine*, Special Issue: Information Visualization in Medicine, 22(2):111-131, (2001)
4. Kundu K., Sessions C. DesJardins M., Rheingans P. Three-dimensional Visualization of Hierarchical Task Network Plans, *Proceedings 3rd International NASA Workshop on Planning and Scheduling for Space,* Houston, TX (2002)
5. Marinagi C., Panayiotopoulos T., Vouros G., Spyropoulos C. Advisor: a Knowledge-Based Planning System, *International Journal of Expert Systems* 9(3):319-355 (1996)
6. Vrakas D., et. Al. Learning Rules for Adaptive Planning. *Proceedings 13th International Conference on Automated Planning and Scheduling.* (2003)
7. Vrakas D., Vlahavas I. A Heuristic for Planning Based on Action Evaluation. *Proceedings 10th International Conference on Artificial Intelligence: Methodology, Systems and Applications.* (2002)
8. Vrakas D., Vlahavas I. Combining Progression and Regression in State-space Heuristic Planning. *Proceedings 6th European Conference on Planning*. (2001)