Combining constraint-based and classical formulations for planning domains: GIPO IV

T. L. McCluskey and R.M.Simpson
School of Computing and Engineering
The University of Huddersfield, Huddersfield HD1 3DH, UK
t.l.mccluskey@hud.ac.uk

November 21, 2006

Abstract

In this paper we compare classical and activity/constraint-based formalisms for formulating planning domains. From this comparison we argue that it is desirable to create an abstract (graphical) notation for describing conceptual models of domains that could be translated to classical and/or formalisms after refinement. This may help to clarify the relationships between different language families. We describe work in progress to create a graphical interface for knowledge acquisition (GIPO IV) that encapsulates this abstract view.

1 Introduction

Deploying software agents that can synthesise courses of action from precise and detailed knowledge in a real, continuously changing environment (such as in the Mars Rover application [4]), is a particularly difficult feat of engineering. The control mechanisms of such applications need to be able to represent and reason with accurate knowledge of such phenomena as movement, resource consumption, and unpredictable environmental conditions. Representation of real world phenomena such as continuously changing processes and uncertainty in a form that can be reasoned with is still a great challenge for AI [3]. In particular, many environmental occurrences are outside the explicit control of plan execution, yet have to be reasoned with during activity generation. Research to automate planning in domains which contain such continuous processes is important and ongoing in several research groups (eg in the Universities of Toronto and Strathclyde [18, 12]). However, there has been little research into the *acquisition and formulation* of such application knowledge.

There are are many well known areas within computer science where formalisation is appropriate, ranging from examples describing the properties of computer hardware using the temporal logic-based standard PSL, to formally recording the requirements of software in Z, to capturing the dynamics of hybrid systems [6]. The importance of the representation languages

for representational adequacy, reasoning, and tool support has been recognised throughout these areas. Languages for domain formulation in planning applications also require these properties: the difference is that for planning, the language must be adequate to be used as a basis for the automated synthesis of plans. Looking at the wider context of encoding of dynamical systems this can be a subtle point: while conventional formulations generally support simulation, in planning the formulations are used as the basis to synthesis another formulation (the plan) which can then be used in simulation.

The importance of knowledge engineering - particularly the acquisition and formulation processes - to the success of real applications of planning has been well documented in the development of space and military applications [8, 9, 13] as well as in more recent semantic web applications [20, 14]. From their analysis of the characteristics of the application area, firstly, developers must choose an appropriate existing knowledge representation language, or try to extend an existing one, or develop one in-house. The problem is that there is a wide range of language families, and it is not clear what languages are appropriate for what applications, and what similarities and differences lie between existing languages. Secondly, the developers are faced with the arduous problem of formulating knowledge in the chosen language. Invariably this would entail the construction of in-house tools and processes to aid this phase of engineering.

Our work is aimed at alleviating the problems described above. To this end we are striving to find abstract graphical notations or visual metaphors expressive enough to aid the formulation of applications involving complex dynamics. We are researching appropriate tools that can potentially translate the graphical notations to an appropriate formulation, given the acquired knowledge and environmental factors. This will help knowledge engineers explore different formulations of a planning domain at an early stage.

In this paper we compare two families of formalisms which represent very different views of the planning problem: domain representation languages for constraint-based planning and scheduling, and for classical planning. For the former, we focus on the formalism called NDDL which is used by NASA Ames for space applications. NDDL and classical planning formalisms typified by PDDL have the same aim: to encode the dynamics of a domain; but they differ in approach, and in syntax and in semantics. Using the competition version of the GIPO III tool [1], we show how it is possible to develop a domain model of the mars rover example using a classical approach. This comparision highlights the strengths of both NDDL and GIPO as tools for planning knowledge formulation. In the following section we introduce GIPO IV, which has been partially implemented as a successor to GIPO's competition configuration. GIPO IV is an attempt to create an interface which a developer uses to encode dynamic knowledge, and from which formalisms such as PDDL or NDDL can be automatically generated as appropriate to the application.

2 Constraint-based Planning and Scheduling: NDDL

Working in the area of space technology, scientists have developed application-oriented language families for knowledge formulation such as HSTS [15] and DDL [7]. NASA Ames evolved their current language NDDL in-house to formulate their specific type of applications. One

encodes a domain in NDDL in an object oriented fashion, with object instances representing persistent actions called entities. These entities naturally form contiguous *timelines* for each object instance, so that at each point in time an object is engaged in some activity (even at rest). Each entity on a timeline is called a *token*, and has a duration (or duration *interval*, to capture the uncertainty of how long actions will take). At the heart of NDDL is the characteristic that there is no explicit difference between *action* and *state*. The difference is effectively blurred, as tokens represent occurring actions, as well as implicitly holding the state of the system. In NDDL one defines classes of objects, where each object instance has its temporal description given by tokens.

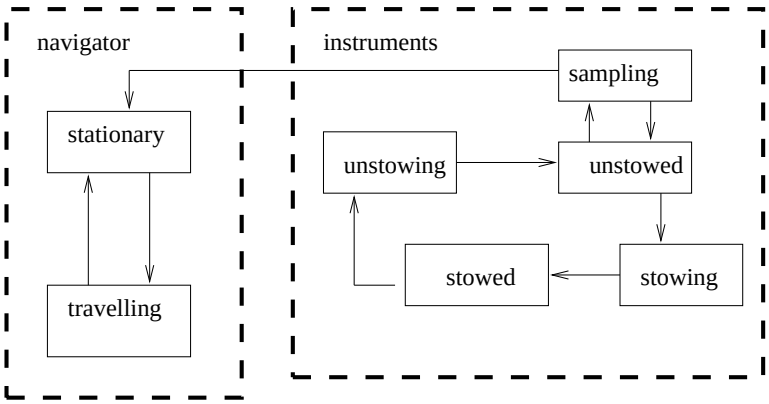

Figure 1: Rover Example in a constraint-based form

NDDL has some attractive features for applications: specifications can be re-used by taking advantage of inheritance and aggregation in the object-oriented sense. Generic objects (like a "resource") can be specialised and used in a particular application (eg if a "battery" were needed to be modelled in a new application, a library class of "resource" could be specialised for this purpose). Similarly the hierarchical composition of objects is effected through a Java-like class system. Using familiar Java-like syntax, a user can also take advantage of user-defined data structures to capture detailed environmental features. Hence, using NDDL the modeller is forced to decompose a domain using the natural form of object life histories. Objects in a domain are described in terms of the typical activities and transitions that they may occupy and undergo. Since activities of objects persist over an interval of time, the idea of concurrent activities is engrained in the formalism.

Tasks in NDDL are specified as a partial plan, and plan generation algorithms take the form of partial-order planning via refinement search. Each cycle of planning involves finding "flaws" in a partially-complete plan, and choosing ways (consistent with the constraints) to complete the plan. Since the timeline idea often removes important choices about inclusion of actions, then planning is mostly about time and resource constraints, which implies that these systems are characterised more as "scheduling" than "planning". Plans are collections of instantiated timelines for the dynamic objects in the application. Possible drawbacks to NDDL is that the syntax is unstructured, with too much apparent freedom being given to the developer. The syntax does not help in clarifying the semantics of the notation, in contrast to a pre- and post condition formulation.

NASA Ames have created tools for plan generation (EUROPA, the planner for NDDL, and

MAPGEN, a configuration of EUROPA specifically for the Mars Rover), and plan analysis (PlanWorks, a tool for inspecting the plan generation process), but there appears to be no similar domain modelling tools in which to manipulate NDDL [1].

To help illustrate and compare NDDL we will use a simple autonomous vehicle navigation domain based on the "mars rover" example which originated from NASA Ames [5]. In this example we formulate the dynamics of a vehicle moving around a landscape and taking samples at specific locations. This entails modelling activities such as moving, sampling, unstowing instruments, as well as modelling environmental concerns such as paths of motion and consequent power use. Figure 1 illustrates the graphics of a simple formulation of the rover in NDDL. There are two timelines ("navigator" and "instruments") showing the temporal constraints within the objects' lifetimes. These constraints are all specified in the domain model using operators based on interval temporal logic. For instance we do not mind what stage the navigator is at when the instruments are being unstowed or stowed, but we do want it to be stationary for the whole period when the sample is being taken. Thus the connection (denoted by an arrow in the diagram) between "sampling" and "stationary" will stipulate that start(stationary) is before start(sampling), and end(stationary) is after end(sampling). The other arrows represent the timeline continuity within the object class eg a typical temporal constraint within a timeline is start(sampling) = end(unstowing). Each of the boxes represents activities and is represented with a predicate. An activity is given a duration (or duration interval) at the modelling stage. Programming-like data types can be used to define resources such as battery charge, and paths for movement. A full formulation of a similar example is available from NASA Ames [5].

3 Classical Planning: GIPO III

Classical planning is based around a world view of domains that are formulated by the declarative specification of parameterised actions. These actions operate on states which represent snapshots of the world being modelled. Tasks are specified by a statement of an initial state, and a set of conditions that much be true of a goal state. It is normally assumed that a controller generates an ordering of instantiated actions to solve some goal posed as state conditions. The idea of discrete states, of actions that change states, of the frame problem resolved by default persistence of state features, are fundamental to classical planning.

PDDL is the medium for exchange of domain models in the classical format through the AI planning research community, being used as a common language in the bi-annual international planning competitions. Over the years the research community have evolved PDDL in an attempt to represent applications with a range of varying environmental concerns. The language has been extended for some real application eg a variant of PDDL is being used in the SIADEX crisis management system [10], and PDDL has recently been extended to represent stream processing for workflow applications [16]. PDDL+ [11] is perhaps the most expressive variant of PDDL including as it does the concepts of event, process, action and continuously changing variables. One can specify mixed discrete/continuous worlds in a language whose semantics is based on hybrid automata. The graphical interface GIPO [19]

[1] Peter Jarvis, NASA Ames, personal communication

enables the acquisition of domain knowledge form a classical planning perspective, and its graphical representations can translate into PDDL or its own internal formalism OCL. We will use GIPO to graphically illustrate a version of the Mars Rover domain, as

- GIPO embodies an object-oriented approach to acquisition, and a life history editor. These make it less distant from the NDDL notation.

- GIPO can represent knowledge with features found in PDDL+.

Figure 2 is GIPO's graphical representation of the simple rover domain using its "object life history editor". It consists of events "arrived", "sampled", "stow_complete", and "unstow_complete" (coloured pink in a coloured version of the paper), and processes indicated with a clock icon. The other curved-corner boxes are actions and coloured green. As with the NDDL version there are two object classes - navigator (with the robot icon) and instrument (with the gripper icon). States of the two object classes are represented by named rectangles, and classes are distinguished by icons (here a robot icon for navigator states, and a gripper icon for instrument states).

The arrows (again various colours are used for differing roles) show the dependency between states, events, processes and actions. In general, a process has been added next to an object state to represent some durative, changing feature that is taking place. This is triggered by an instantaneous action, then terminated by an instantaneous event. The diagram appears more complex that the illustrative NDDL diagram: this is partly because "activities" are more awkwardly represented in the classical notation; but also because GIPO translates the graphics into a domain specification, and hence needs more details to produce the formulation. Additional to the graphics, a developer must specify fluents that are affected by processes and that trigger events, by entering formulae representing the continuous changes.

4 Comparison of the Rover Formulations

The OO nature of GIPO allows a life history of the navigator and the instrument classes to be written in a similar form as NDDL. The sequences of transitions that objects of these classes pass through is made explicit. The "active" activities in NDDL are represented in GIPO using an action to start the activity, a state to represent an object being engaged in that activity, a process to simulate the continuously changing properties (resource and/or movement), and an event to represent the end of the activity and the start of a new one. "Inactive" activities such as "stowed" and "stationary" are represented as single states. This appears more awkward in this particular classical formulation, although what a rover control can plan for is made explicit by the distinction between actions and events. In NDDL this distinction is implicit. The temporal constraints in the GIPO version almost capture the required constraints between activities in the example: the link from "stationary" state to the "take_sample" action and "sampling" process guarantee that during sampling the navigator must remain stationary. However, the modality of this would not dissallow plans which moved the navigator during sampling - this would entail that sampling would immediately stop.

The overall advantage of GIPO is that, given that appropriate constraints are attached to

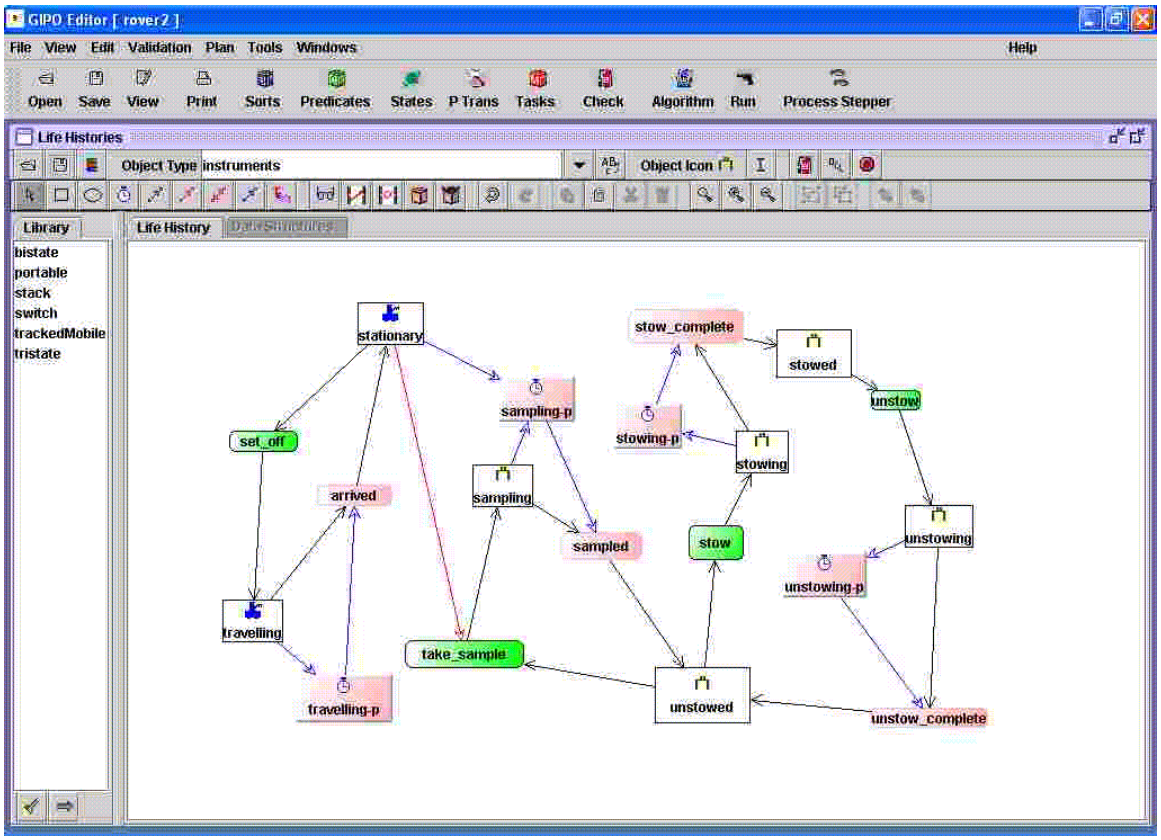

Figure 2: Rover Example in GIPO III

the continuous variables in each process and event node, the system can first test for the consistency of the model, simulate the operation of the model using GIPO's "stepper", and then generate OCL+ (an OO version of PDDL+) for output to a planner. However, GIPO's embodyment of classical planning does not allow for uncertainty of resource consumption, full interval temporal constraints or user-defined data structures. From this comparison, it appears that a graphical interface would be useful that enables the developer to generate a high level description of the domain independent of which type of formalism.

5 An interface to capture conceptual dynamic information: GIPO IV

GIPO IV is a partially implemented predecessor to GIPO III which inherits its ideas and techniques of automated domain model generation from diagrammatic representations, but further exploits the ideas of object-orientation in formulations and provides abstract views of a domain independent of the intended target formalism. With GIPO III, domain models could be generated in PDDL or in the native OCL language. In version IV, we are working towards the generation of models (additionally) in an activity-based notation like NDDL. The idea is that the graphical notation provides a level of abstraction higher than of the

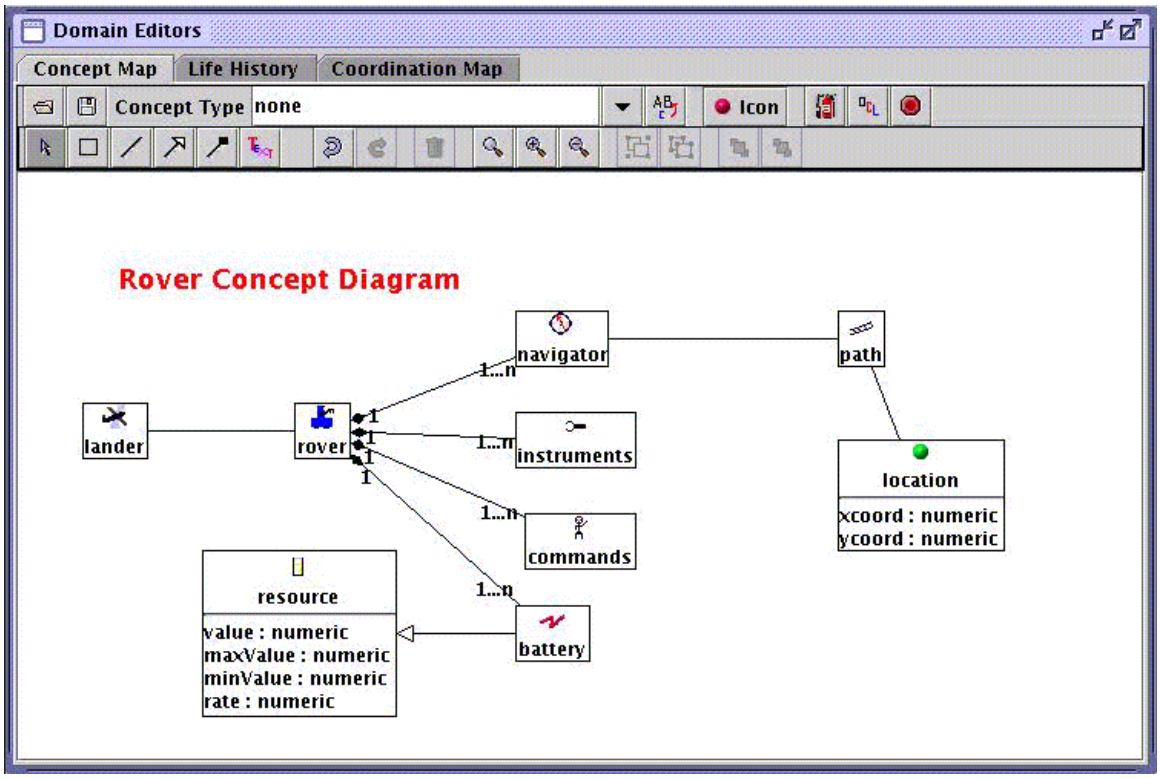

Figure 3: The Concept Diagram in GIPO IV

particular syntax chosen. The developer will be able to adjust environmental conditions and have a choice of formalism generated. This should give a route to comparing the capabilities of different planning systems using the same conceptual model of a domain.

In GIPO IV a developer designs a planning domain model using a concept diagram in the tradition of object-oriented design. Such a concept diagram for the rover example is shown in Figure 3 [2]. The concept diagram models the whole system as an aggregation of components, with a specified relationship between the system and each component. Each subcomponent can have attached attributes made up of user-defined data-types. In the mars rover example, a rover is made up of the four types of sub-component specified. In OO-style, the relationship is annotated with its cardinality (in this case 1 to many). The navigator has a path attribute as specified, and the battery inherits a state from a (pre-defined) resource component.

The main function in decomposing the concept diagram is to specify the life histories of the objects that are instances of each of the components (although some components may not, as in the case of the battery, need to be represented with a life history). In GIPO IV we create these using a separate *life history diagram* for each component. Here, each "activity" is encapsulated in a node, with arrows representing the next possible activity (see Figures 4 and 5). These diagrams are similar to the life history diagrams of GIPO III, with extra information: stop and start states can be specified, to give an advanced idea of the kind of task information expected, and attribute information can be included in the node. On the

[2] this is actually more detailed than the previous examples, and reflect more faithfully NASA's rover example

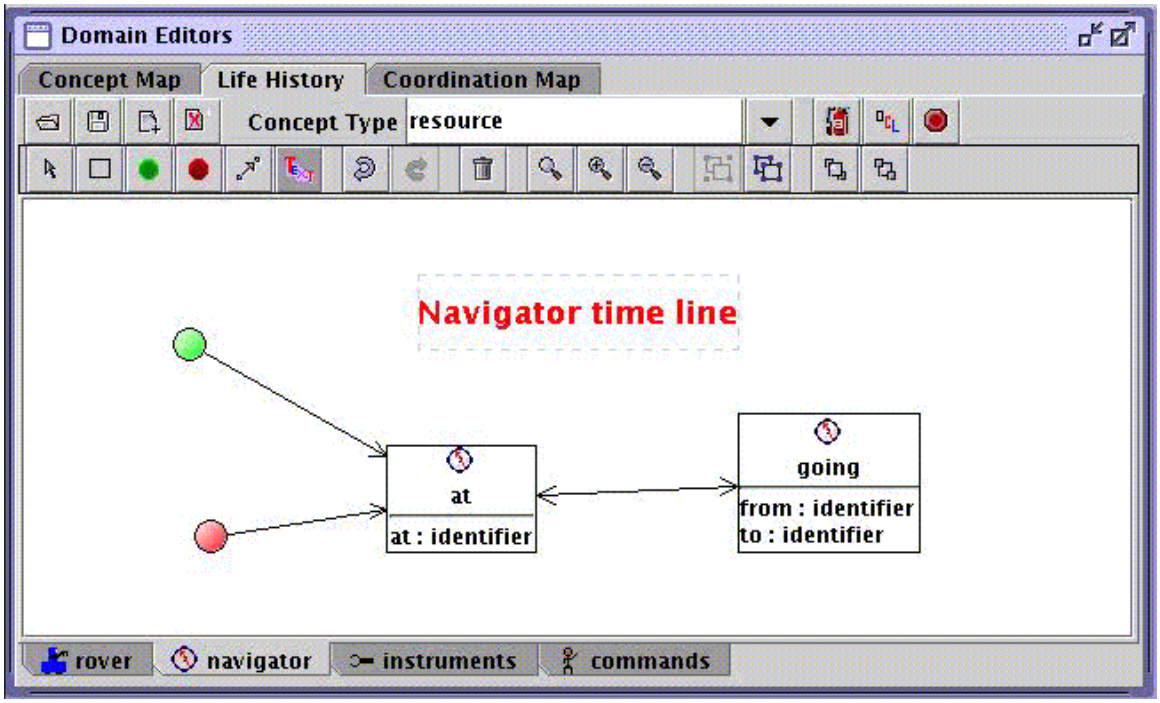

Figure 4: The Life History of the Navigator

other hand, we suppress interval or durative information: the arrows can be interpreted at this stage as instantaneous activity change. Figure 4 specifies the timeline of a navigator, while Figure 5 specifies the timeline of the rover's instruments. These stipulate that the normal stop and start state for a instrument is "stowed", and the normal stop and start state for a navigator is in the rest position of "at".

To include temporal information and constraints for connected components, the developer creates a set of *co-ordination maps*. These diagrams are used to specify the dependencies between objects of different classes. In the example in Figure 6 temporal constraints are placed on the takesSmple node (the arrow is coloured red) to stipulate that sampling may only take place during the "at" activity.

At this stage of domain development, the system has acquired information within conceptual maps (defining the object class structure) within life history diagrams (describing activity transitions), and within connection maps (describing the depedencies between life histories). It is possible to generate *both* a NDDL formalisation, and a classical formulation of the problem domain, and that this would be useful to experiment with different formulations and planning engines.

6 Conclusions

Modelling for simulation, in such areas as timed automata for real time systems [2], or hybrid automata for industrial plant or transportation control [6], relies critically on the use of tools

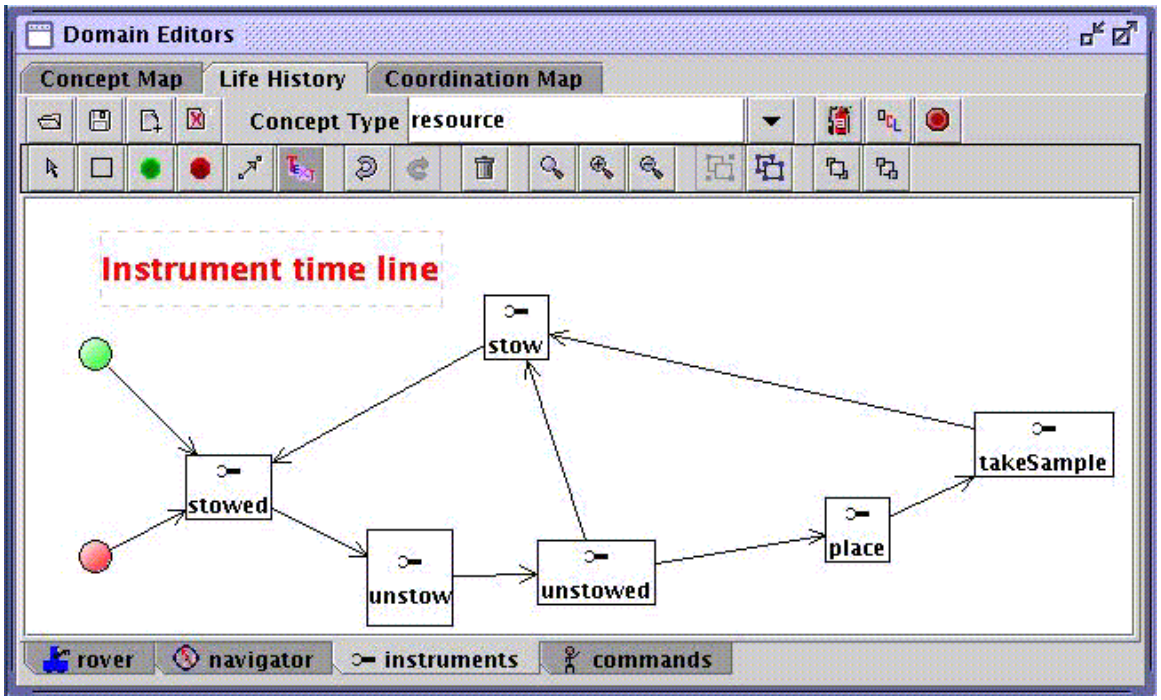

Figure 5: The Life History of the Instruments

for engineering simulation models. Where automated planning differs is that we are modelling for reasoning, synthesis, then simulation. It therefore appears that engineering tools are even more crucial in automated planning than these more established, related disciplines. Reiter, in the Preface to his textbook [17] stated "When faced with a dynamical system that you want to simulate, control, analyse or otherwise investigate, first axiomatise it in a suitable logic". While a direct axiomatisation is very desirable and comes with all the benefits advocated in his book, the drawbacks of this approach are that an axiomatisation within the fluent or situation calculus is not within the capabilities of all developers (considering the difficulties in formulating a set of successor state axioms [17]) and is not appropriate for all applications. Also, the problems of acquisition, maintenance and knowledge re-use are not addressed by his approach. The problem is the difference between engineering and theoretical enquiry; which Reiter's quote fits the the latter, it is inadequate for the former. Hence we would amend the quote to read ".. first create an abstract conceptual model using an appropriate GUI, then explore the additions of environmental constraints, and then use the GUI to automatically translate the knowledge acquired into a formulation for an appropriate planner".

According to the authors of NDDL, the 'classical' approach to automated planning takes the over-simplistic view of planning as a set of fluents whose values are changed as each step / action. Apparently, this is too simplistic for Space applications. It appears to us that the most significant drawback of classical planning is that it reflects the concerns of those working in *generative* planning, rather than more execution and scheduling orientation of many applications. We argue here for an abstract formualtion of planning that is (initially) neutral to the underlying semantics of the target formalism. Rather, the abstract view can be used as a basis for a translation to a number of formalisms. In the near future We plan to

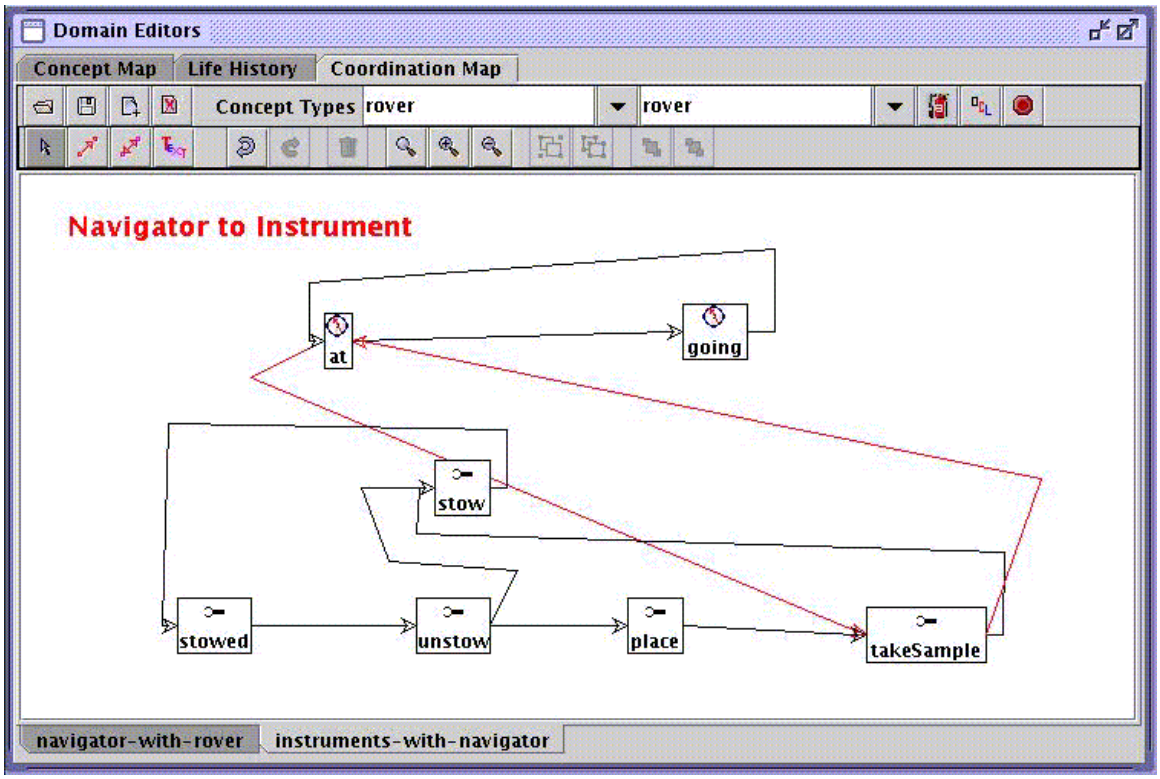

Figure 6: Rover Example in GIPO IV

complete development of GIPO IV and provide demonstrable proof of this assertion.

Ackwledgement

The authors would like to thank Peter Jarvis of NASA Ames for his comments on an earlier version of this paper.

References

[1] R. Bartak and T. L. McCluskey. The First Competition on Knowledge Engineering for Planning and Scheduling. *AI Magazine*, 2006.

[2] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets*, pages 87–124, 2003.

[3] John L. Bresina, Richard Dearden, Nicolas Meuleau, Sailesh Ramkrishnan, David E. Smith, and Richard Washington. Planning under continuous time and resource uncertainty: A challenge for ai. In *UAI*, pages 77–84, 2002.

[4] John L. Bresina, Ari K. Jónsson, Paul H. Morris, and Kanna Rajan. Activity planning for the mars exploration rovers. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005), June 5-10 2005, Monterey, California, USA*, pages 40–49, 2005.

[5] C. McGann. Problem Solving in Europa 2.0. Technical report, NASA Ames Research Centre Technical Report, 2006.

[6] Luca P. Carloni, Roberto Passerore, Alessandro Pinto, and Alberto Sangiovanni-Vincentelli. Languages and tools for hybrid systems design. *Foundations and Trends in Design Automation*, 1(1):1–204, January 2006.

[7] A. Cesta and A. Oddi. DDL.1: A Formal Description of a Constraint Representation Language for Physical Domains. In M. Ghallab and A. Milani, editors, *New Directions in AI Planning*, pages 341–352. IOS Press, 1996.

[8] S. A. Chien. Towards an Intelligent Planning Knowledge Base Development Environment. In *Planning and Learning: On to Real Applications. Papers from the 1994 AAAI Fall Symposium*, number FS-94-01. AAAI Press, 1995.

[9] S. A. Chien. Knowledge Acquisition, Validation, and Maintenance in an Automated Planning System for Image Processing. In *Proceedings of the Tenth Workshop on Knowledge Acquisition for Knowledge-based systems, Banff, Canada*, 1996.

[10] Juan Fdez-Olivares, Luis Castillo, Oscar Garcia-Perez, and Francisco Palao Reins. Bringing users and planning technology together. experiences in siadex. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006), June 5-10 2006, Cumbris, UK*, pages 11–20, 2006.

[11] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains . In *Technical Report, Dept of Computer Science, University of Durham*, 2001.

[12] Richard Howey, Derek Long, and Maria Fox. Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In *ICTAI*, pages 294–301, 2004.

[13] J.K. Kingston, A. Griffith, and T.J. Lydiard. Multi-perspective modelling of the air campaign planning process. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 668–677, 1997.

[14] E. Martinez and Y. Lesprance. Web service composition as a planning task: Experiments using knowledge-based planning. In *Poceedings of the ICAPS-2004 Workshopon Planning and Scheduling for Web and Grid Services, 62-69, Whistler, BC, June, 2004*, 2004.

[15] N. Muscettola. HSTS: Integrating planning and scheduling. In *Intelligent Scheduling*, pages 169–212. Morgan Kaufmann, 1994.

[16] nton Riabov and Zhen Liu. Scalable planning for distributed stream processing systems. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006), June 5-10 2006, Cumbris, UK*, 2006.

[17] Raymond Reiter. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems.* MIT Press, 2001.

[18] Ji-Ae Shin and Ernest Davis. Processes and Continuous Change in a SAT-based Planner. *Artificial Intelligence*, 166, 2005.

[19] R.M. Simpson. Gipo graphical interface for planning with objects. In *Proceedings of the International Competition for Knowledge Engineering in Planning and Scheduling*, Monterey, 2005.

[20] A. ten Teije, F. van Harmelen, and B. Wielinga. Configuration of web services as parametric design. In E. Motta, N. Shadbolt, A. Stutt, and N. Gibbins, editors, *Proceedings of the 14th International Conference, (EKAW-2004)*, number 3257 in Lecture Notes in Artificial Intelligence, pages 321–336, Whittleburry Hall, UK, October 2004. Springer Verlag. ISBN 3-540-23340-7.