Supporting Plan Authoring and Analysis

Jihie Kim and Jim Blythe
University of Southern California/Information Sciences Institute
Marina del Rey, CA 90292 USA
+1 310 448 8769
{jihie,blythe}@isi.edu

ABSTRACT
Interactive tools to help users author plans or processes are essential in a variety of domains. KANAL helps users author sound plans by simulating them, checking for a variety of errors and presenting the results in an accessible format that allows the user to see an overview of the plan steps or timelines of objects in the plan. From our experience in two domains, users tend to interleave plan authoring and plan checking while extending background knowledge of actions. This has led us to refine KANAL to provide a high-level overview of plans and integrate a tool for refining the background knowledge about actions used to check plans. We report on these lessons learned and new directions in KANAL**.**

Categories and Subject Descriptors
H.5.2 [**Information Interfaces and Presentations**]: User Interfaces – *User interface management systems*

General Terms
Verification, Human Factors

Keywords
knowledge acquisition, plan analysis, knowledge bases, plan authoring, process models.

1. INTRODUCTION
End user programming is a key research area in developing intelligent user interfaces. There has been a wide range of approaches taken, including programming by demonstration [8] and learning apprentices[16]. These systems make use of observed examples and generalize them into a task representation that can be used in the future. While these approaches work well for simple tasks, as the complexity of the knowledge to be captured increases, direct knowledge authoring tools seem more useful.

In the past we have built various knowledge editors and tools to enable end users to specify new knowledge directly [2,12,3].

Recently, we have been investigating approaches to helping end users author and analyze process models and plans, including process models in biology and plans in military applications. Often process models are quite complex, as shown in Figure 1. The figure illustrates a part of a process model for a virus invading a cell, entered by a graphical knowledge entry tool [6]. These process models can have many steps and objects that are connected by a variety of links. For example, there are different types of connections among the steps, including decomposition links between steps and substeps and ordering constraints. Objects are connected to the steps based on the roles they play. Helping end-users author and check such complex plans and process models remains a challenging area.

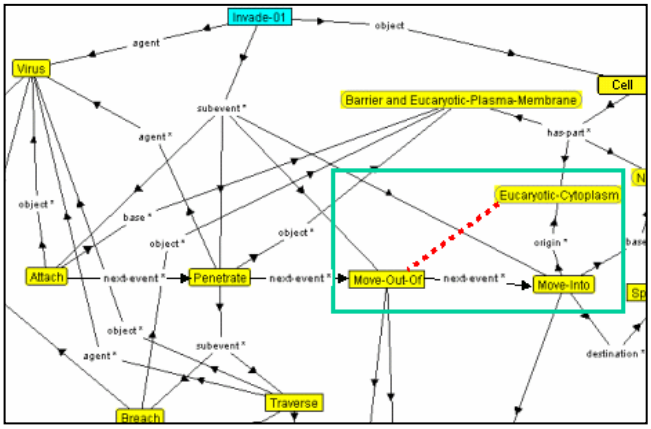


Figure 1: Example process model: a virus invading a cell.

Existing approaches for supporting plan authoring include graphical tools that let users layout plans by drawing steps and connections between the steps. The help provided by these tools is limited because there is no underlying semantics that can support the analysis. There are plan analysis approaches that defined some desirable properties of plans and some use critics that detect problems [10]. Also there are tools that capture planning knowledge interactively [5,17] but they are not designed to exploit background knowledge and ontologies, which we believe is crucial technology to providing strong guidance needed by end users.

KANAL (Knowledge ANALysis) is a tool that checks process models specified by a user, reports possible errors and provides specific suggestions to the users about how to fix those errors [13]. It helps users build or modify process models by detecting

invalid statements and pointing out what additional knowledge needs to be acquired and what existing knowledge needs to be modified. These are done by analyzing interdependencies between different pieces of knowledge used in describing the process model and also relating them to the existing knowledge and ontologies. KANAL is currently used within an end-to-end knowledge acquisition (KA) system called SHAKEN [7]. Users can invoke KANAL whenever they want to check their definitions of process models. SHAKEN has general background knowledge on actions that KANAL makes use of in analyzing process models. For example, conditions and effects of a general Move concept can be used in analyzing any Move steps in a process model.

KANAL has been used by various end users including biologists who built and tested models of complex molecular biology processes and army officers who used KANAL to critique their plans, i.e., courses of action (COAs). We found that its analysis report was very useful for checking the plans that were entered by users and often suggested useful ways of improving the plans. However we also found that sometimes users wish to extend the background knowledge that is used to produce the analysis report. For example, they may want to define various special cases of actions that are relevant for different situations, or exceptions that were not specifically addressed in the general background knowledge. In general, knowledge bases are never complete and it is important to provide a capability of adapting the knowledge base for varying needs. To support this capability we have extended our authoring environment so that the user can choose to enter special cases of critiquing knowledge as well as to change the description of the process itself in order to improve the resulting report. In this extension, instead of describing different cases in a single definition, we make use of the inheritance mechanism in the given knowledge representation system to represent each special case as a separate entity. We argue below that this approach provides a more a natural view of special cases for end users and has efficiency advantages.

In this paper, we report our experience in helping end users author and analyze process models in two different domains and present how we improved our interfaces based on the experience. We begin with a brief overview of KANAL through an example of the checks it performs and then we describe how the KANAL report was used by end users during a sequence of two evaluations. Next we describe the feedback we have received from end users and how we improved the KANAL interface based on the feedback. Then we describe an extension that allows users to interleave plan authoring and plan checking by creating special cases of actions in order to improve critique results. Finally, we discuss our future work in developing additional support for plan building and checking.

2. KANAL: Helping users analyze authored plans

The current implementation of KANAL is built on the KM knowledge representation and reasoning system [6]. KM provides a function that can execute a step in a given situation and create a new situation based on the effects (add/delete list) of the given step. KANAL uses this function to simulate the given process model and analyzes interdependencies between the conditions and effects of the steps, such as that the required conditions for each step are met when the step is supposed to take place, and that the expected effects of the overall process are in fact obtained. KANAL also checks how different steps are related to each other, including their temporal ordering and causal relationships. In the process, KANAL reports possible errors in the models, and generates specific suggestions to the user about how to fix those errors.

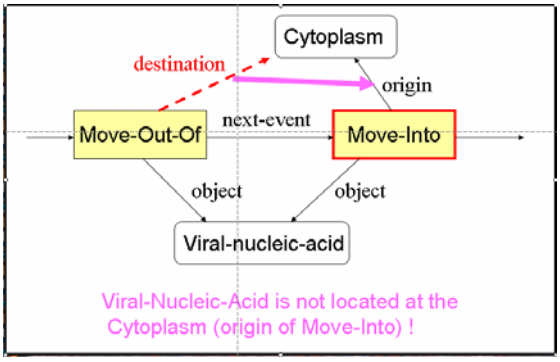


Figure 2: A missing role assignment (missing "destination" link between Move-Out-Of and Cytoplasm) is detected through a failed precondition of the following step.

The highlighted box in Figure 1 shows an example mistake while building a model of a virus invading a cell. Currently, the destination of the Move-Out-Of step is missing. The editor prevents users entering invalid links based on the domain and the range of the roles, but such missing information needs more thorough analysis of the dependencies between the steps. KANAL notices this problem when it checks the precondition of the next step (Move-Into), i.e., the Viral-Nucleic-Acid needs to be located at the Cytoplasm in order to move from the Cytoplasm (the origin). The precondition fails because the location of the Viral-Nucleic-Acid is currently unknown due to the missing link. If the Move-Out-Of step had such a link, it would have made the location of the Viral-Nucleic-Acid as the Cytoplasm, as shown in Figure 2. For this type of problem, KANAL's proposed fixes include 1) adding a new move step to make the location of the Viral-Nucleic-Acid be the Cytoplasm 2) modifying a previous Move (Move-Out-Of) to make the destination be the Cytoplasm or 3) modifying the current step so that the condition no longer needs to be satisfied. In fact, the second suggestion directly points to the user's mistake: missing destination link between Move-Out-Of and Eucaryotic-Cytoplasm.

Besides failed preconditions and missing links, KANAL can check unnecessary links, undoable steps, useless steps, enabling relationships among steps, unachieved expected effects, disjunctive branches, loops, etc. The details are available in [13].

Table 1 shows the number of times KANAL was used during a sequence of two DARPA Rapid Knowledge Formation (RKF) program evaluations. Four biologists participated in the first evaluation and three of them participated again in the second, smaller scale evaluation. In both evaluations, the biologists created models of processes described in a cell biology textbook. The quality of the resulting knowledge base was assessed with a set of textbook type questions.

	Summer 2001	January 2002
Total number of concepts built	449	157
KANAL invocations	144	71
Invocations per concept	**0.32**	**0.45**

Table1: Uses of KANAL

The first row shows the number of concepts built by the biologists and the second row shows the number of KANAL invocations during the evaluations. Between the two evaluations, it seemed that as they become more familiar with the tool, the biologists used KANAL more frequently (from 32% to 45% of the concepts). Note that the concepts created by the users through SHAKEN included very simple factual definitions that didn't need to be checked by KANAL. For example there were many simple subclass definitions, such as a definition of Cap as a subclass of DNA-Sequence.

Error/warning Type	Summer 2001		January 2002	
	Total #	ratio	Total #	ratio
Missing first-event, subevent, next-event	37	0.26	8	0.11
Unreached events	55	0.38	16	0.23
Unnecessary ordering	105	0.73	52	0.73
Failed conditions	133	0.92	111	1.56
Failed execution of step	30	0.21	24	0.34
Effectless step	139	0.97	6	0.08
Failed expected effect	7	0.05	10	0.14
Loop	1	0.01	0	0

Table2: Errors and warnings reported

(ratio: number of errors or notes / number of KANAL invocations)

Table 2 shows the number of errors and warnings reported to the users during the evaluations. As shown in the table, KANAL was used in performing various types of checks, including missing links, failed conditions, failed executions, etc. The changes in the ratios between the two evaluations are related to multiple different factors. For example, there were fewer warnings on simple errors like missing event links (first-event, subevent, and next event links) and unreached events. It seems that as the users become more experienced in building process models, they tend to make fewer such mistakes. However, there were some other changes between the two evaluations. For example, SHAKEN's background knowledge has been improved over time and this improvement affected the KANAL reports as well. As shown in the table, KANAL detected more failed conditions per invocation in the second evaluation because there were more action conditions defined in the background ontology which KANAL can make use of to perform more thorough checks. On the other hand, there was a significant reduction in the number of effectless steps as there were more effects defined for each action, which reduces chances that a user may define a step that doesn't produce any effect.

In summary, end users were able to use KANAL to check their process models, and KANAL has reported various problems in their models. During the evaluations, we have received detailed feedback on our interfaces through questionnaires and interviews as well as built-in feedback functions. The following two sections describe what we learned from the users and the improvement we have made based on them.

3. IMPROVING THE PLAN ANALYSIS REPORT

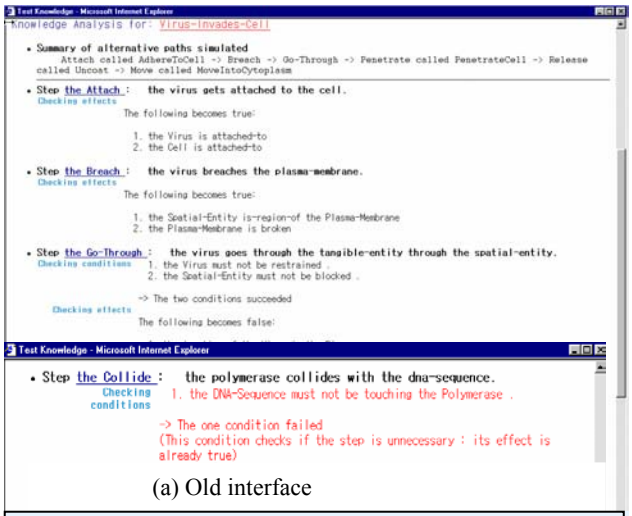
From the interactions with the users, we learned some important ways of making the plan analysis report more useful. In this section we summarize what we have learned from our experience and the improvements we have made. We believe that these results may be useful for other plan editing and analysis tools.

• *Helping users understand results from complex analysis:*

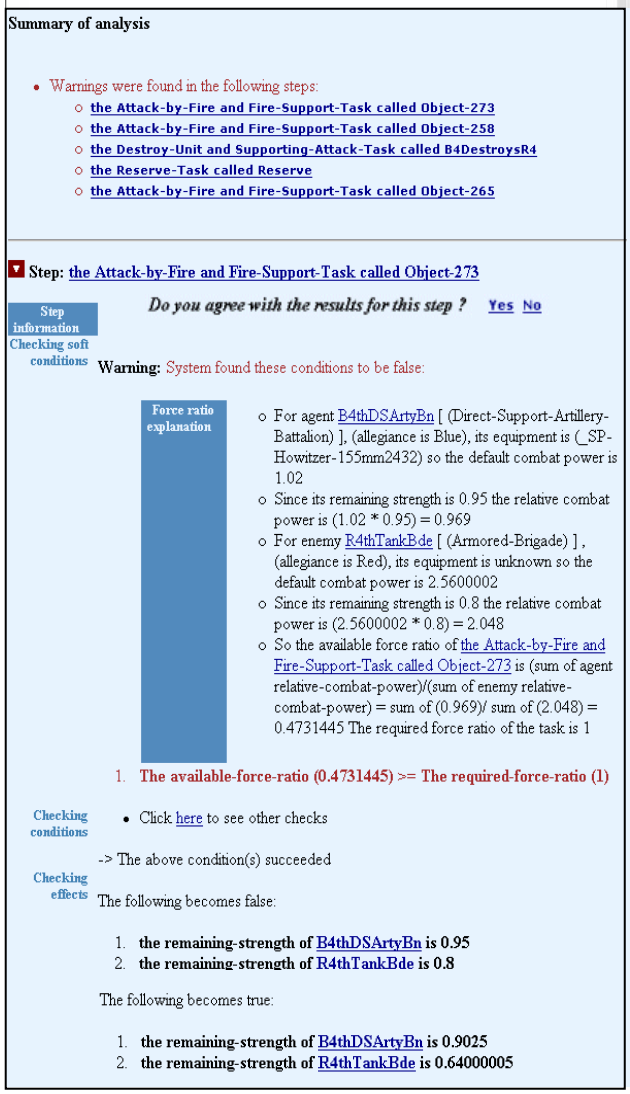
Figure 3-(a) shows a part of the old KANAL interface that reports warnings found from a model. The users were not able to examine how the checks were done and why the condition was not achieved, which made the KANAL report hard to understand. Many knowledge-based systems suffer similar problems in presenting problem solving results and simply showing the proof trace of the problem solver doesn't help users much. In most cases, the traces usually contain a lot of details users don't understand, although some KR languages such as EXPECT support a more natural way of explaining problem solving traces [2].

Figure 3-(b) shows the current KANAL interface that presents a critique of a COA built by an Army general. It has a warning that one of the preconditions failed, i.e., the given combat power ratio is not high enough to perform the given task. The explanation section in the middle shows how combat power analysis is done by combining various pieces of information, including unit equipment, the default combat power, remaining strength of the units, etc., through multiple steps of computation. Users found this addition very useful and the explanation seemed to make the results more reliable. Currently KANAL uses templates that can summarize the objects that participate in the analysis and the steps involved but we are planning to investigate a more general way of supporting this function such as a way of employing more explainable language.

• *Providing a focused report*:



(a) Old interface



(b) The new interface explains why the condition has failed.

Figure 3: improving KANAL interface

Many of KANAL's checks make use of SHAKEN's background knowledge on actions [1] which include knowledge about general action types (such as Move) and more special knowledge that is needed for checking domain dependent conditions and effects. For example Move in general requires that the object being moved shouldn't be restrained and the objects involved should be known but in moving military units, one should also consider if the assigned military equipment is able to traverse the given terrain. Although general knowledge about actions is useful in many cases, we found that users often want to focus on only the domain dependent checks before moving on to other checks provided from general knowledge. This seems especially true when the users have given little information that is needed to perform the general checks. For example military officers don't explicitly state if the object is restrained or not when they build COAs.

Currently KANAL provides a focused view by creating a boundary between the checks that are more related to the topic and other general checks. However a more general way of supporting different views of the same results is desirable. For example, the system may make use of the context and the user's interests to filter out unnecessary details. In Figure 3-(b), KANAL shows only focused checks but users can see more details by clicking "Click here to see other checks".

• *Providing a summary of state transition*:

remaining-strength

Unit Names	Initial Values	the Fire-Support-Task and Atk-by-Fire-on-MechInf called Object-282	the Fire-Support-Task and Atk-by-Fire-on-MechInf called Object-281	the Fire-Support-Task and Atk-by-Fire-on-MechInf called Object-280	the Fire-Support-Task and Atk-by-Fire-on-MechInf called Object-279
B1stGSArtyBn	0.95	0.9	0.86	0.81	0.77
R3rdTankBde	0.8	0.64	0.51	0.41	0.41
R2ndTankBde	0.8	0.64	0.51	0.51	0.51
R2ndBn1stBde	0.8	0.64	0.51	0.41	0.33
R3rdBn1stBde	0.8	0.64	0.51	0.41	0.33
R29thFieldArtyRegt	0.7	0.56	0.56	0.56	0.56
B2ndAVNBn	0.95	0.95	0.95	0.95	0.95
B1stAVNBn	0.95	0.95	0.95	0.95	0.95
R4thTankBde	0.8	0.8	0.8	0.8	0.8
B4thDSArtyBn	0.95	0.95	0.95	0.95	0.95
B1stDSArtyBn	0.95	0.95	0.95	0.95	0.95
B2ndDSArtyBn	0.95	0.95	0.95	0.95	0.95
B4thTankBde	0.94	0.94	0.94	0.94	0.94
R1stBn1stBde	0.8	0.8	0.8	0.8	0.8
B3rdMechInfBde	0.95	0.95	0.95	0.95	0.95
B2ndTankBde	0.94	0.94	0.94	0.94	0.94
B1stTankBde	0.94	0.94	0.94	0.94	0.94
B23rdCavSqdn	0.95	0.95	0.95	0.95	0.95
B3rdDSArtyBn	0.95	0.95	0.95	0.95	0.95

location

Unit Names	Initial Values	the Fire-Support-Task and Atk-by-Fire-on-	the Fire-Support-Task and Atk-by-Fire-on-	the Fire-Support-Task and Atk-by-Fire-on-	the Fire-Support-Task and Atk-by-Fire-on-	the Fire-Support-Task and Atk-by-Fire-on-	the Attack-by-Fire and Fire-Support-	the Attack-by-Fire and Fire-Support-

Figure 4: summary of time varying properties

KANAL analyzes process models by performing a simulated execution of the steps to find what effects and changes were made by each step. For example, in simulating military actions of a given COA, the unit strengths will be reduced over time depending on the engagement tasks the units were assigned, and the unit locations will be changed based on the move steps involved. While examining the KANAL output, users had difficulty in understanding such transitions over time, and they suggested that we provide a way of summarizing the changes made to the objects through each step. Other planning tools have also found it useful to provide a visualization of changes made to objects during a plan [19].

The current implementation of KANAL supports this by generating a table for each property that changes over time. For example, for the COA domain, it generates a set of tables that show how the unit strength changes over a sequence of tasks, how the unit location changes, etc, as shown in Figure 4. This kind of table seems to be more useful than the textual description of steps KANAL used before, because the tables can show the overall transitions more concisely than a text listing of the changes made by each step.

- *Getting user feedback*:

Users may agree or disagree with the checks made by the system. Whenever there is a disagreement it could be because 1) the plan they built is different from what they intended, 2) the background knowledge used to check their plans is inconsistent with their knowledge, 3) the KANAL results are not clearly presented in the interface, or 4) the system has other bugs. To be able to check if the results are in fact consistent with their expectations, we have built a way of getting user feedback by adding a couple of more links in the interface. As shown in the step heading in Figure 3-(b) ("Do you agree with the result for this step?"), users can indicate if they agree or disagree and describe the reasons when they disagree. We found that this kind of user feedback is very useful for understanding what worked and what didn't, and directs ways of improving our system further.

4. THE ACTION EDITOR

The process models that are checked in KANAL are hierarchically organized collections of actions represented in a STRIPS-like language, similar to that used by many AI planners [4,11]. As users gain a better understanding of a process using KANAL's plan checker, they may wish to change either the process model or KANAL's knowledge about the actions including their preconditions and effects. In this section we describe the Action Editor that allows users to change knowledge about actions. This ability is crucial for users to feel that the process checking component of KANAL is giving them useful information and is also a challenging UI and knowledge acquisition task.

4.1 Action special cases

As mentioned above, SHAKEN's knowledge of actions includes general actions and their behaviors, such as **move**, and domain-dependent actions, which are specializations of the general actions. We found that normally, users don't want to change the general action definitions, which are quite stable, but they often wish to add more detail to an action to cover slightly different behavior in special cases. If these special cases were represented in a normal STRIPS-like syntax, each variation to an action would need to be reflected in one set of preconditions and conditional effects describing the general action. This can quickly lead to an action definition that is difficult for a knowledge engineer to understand and almost impenetrable to an end user.

Instead, we make use of inheritance in KM, the knowledge representation language used in SHAKEN, to represent each action special case as a separate entity. This approach has several advantages. First, when viewing one of these special cases, the user can see how the action will behave in this particular scenario without reference to all the other special cases. A hierarchical view of the action definitions can show the other special cases that the system can reason about. Second, there is evidence [18] that people find logical statements easier to understand when presented in terms of a general case and its exceptions. Third, we believe that such a representation also has efficiency advantages for a planning system, and similar approaches have also been used for acquiring and representing action knowledge in plan generation systems [19,15].

In order to define a new action special case to KANAL, the user must specify (1) the situations under which the special case applies and (2) the behavior that is different from the normal case of the action. This is done through a graphical editor called from KANAL's process checker. The editor is SHAKEN's concept map editor [7], modified to allow users to enter the situations that define the special case, as we describe in the next section.

4.2 Example

Consider the 'penetrate' step in the scenario of a virus invading a cell described in the first section. SHAKEN includes a generic definition of the **penetrate** action in its initial library, in which an object breaches a barrier and then moves through it. To use this action in the cell biology domain, new preconditions and effects need to be specified when the barrier is a cell membrane and the object is a complex molecule or virus. For example, perhaps the object needs to bond with the outer cell wall, and the time taken or the probability of success may depend on the size of the molecule or virus. Finally, the action may need to be further refined to properly model the behavior of a particular virus when invading a cell, for example to specify that a certain enzyme should be present or may be suppressed in order for the process to work, and that only the viral DNA is inserted, rather than the whole virus.

This process of successive refinement can be captured intuitively by two layers of specializations of the **penetrate** action, the first to refine the generic action for the cell biology domain and the second to capture the specifics of a certain virus. These layers are illustrated in the upper box in Figure 5. Contrast this with a flat, STRIPS-like representation of the penetrate action, shown in schematic form in the lower box in Figure 5. The case for the specific virus and for cells must be separately modeled in the preconditions as well as the generic case. Conditional effects also

need to be specified and the generic effects may need to be modified. When several different virus types are to be modeled, and possibly several different domains, the flat action representation can quickly become unmanageable.

```
operator penetrate
 preconditions (near ?agent ?object)
 effects (and (breached ?object)
              (inside ?agent ?object))

operator penetrate-cell extends penetrate
  use-when (and (virus ?agent)
                (cell-wall ?object))
  additional-preconditions
          (attached ?agent ?object))
  additional-effects
          (not (attached ?agent ?object))

operator lambda-virus-penetrate
  extends penetrate-cell
  use-when (lambda-virus ?agent)
  additional-effects (time-taken 100)
```

```
operator penetrate-with-all-modifications
 preconditions
  (and (near ?agent ?object)
       (or (not (virus ?agent))
           (not (cell-wall ?object))
           (attached ?agent ?object))
       ...)
 effects
  (and (breached ?object)
       (inside ?agent ?object)
       (if (and (virus ?agent)
                (cell-wall ?object))
        (not (attached ?agent ?object)))
       (if (and (lambda-virus ?agent)
                (cell-wall ?object))
           (time-required 100))
       ...)
```

Figure 5: A layered action representation based on special cases is shown in the upper box, and part of an equivalent flat action representation is shown in the lower box.

In KANAL, then, this situation is modeled with a *special case* of the **penetrate** action that pertains to cell biology, with some new or modified preconditions and effects, and a further special case of this **penetrate-cell** action to capture a particular virus. The situations under which the special case should apply are captured by the 'use-when' field as shown in Figure 5, and the changes to the action are captured in the additional-preconditions and additional-effects. Figure 6 shows SHAKEN's graphical editor being used to specify the **penetrate-cell** special case. The agent and object are roles of the action, and they are shown to be a virus and cell wall respectively. These nodes are outlined in green because the user has designated them as defining the 'use-when' condition of the special case. The 'attach' sub-event is not present in the generic version of the action.

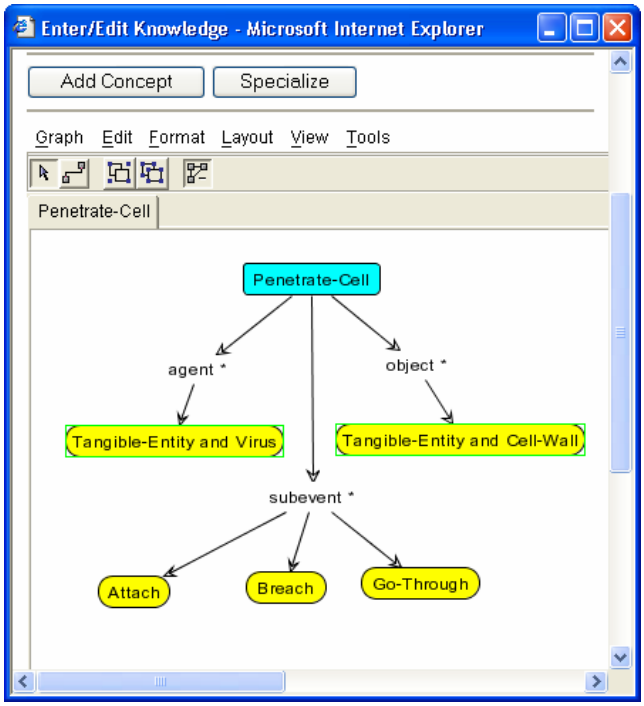


Figure 6: The graphical editor is used to define a special case of the **penetrate** action for cell biology.

4.3 The generality of special cases

The previous example showed how using special cases can help a user to modify action knowledge for checking processes in the domain of cell biology. The special case mechanism was also used in analyzing COAs, where domain experts modified KANAL's knowledge of military actions based on the kind of equipment being used. Initial indications are that they found it quite natural, as we describe below. In addition we have had experience modeling several large planning domains in which we believe using special cases of actions would have made the representations easier for a domain expert to understand and modify. We discuss two such domains.

In the first, the planning task is to recover from damage to a bridge across a river, either by repairing the bridge or by installing a temporary bridge across a nearby stretch of river to allow traffic to cross. There are several different kinds of temporary bridges that can be used, some floating and some unfolded directly from vehicles. These bridges share many of the same requirements for their use, for example in the preparation of the area before emplacement, but they also have many slight variations in requirements, such as the maximum or minimum

river width for which they can be used, and variations in their effects. Therefore we expect that action special cases would greatly help end users in adding or modifying operators for this domain. We see similar patterns in a planning domain for cleaning up after an oil spill at sea developed at SRI, where again the many different kinds of boats and floating booms that can be used have some broad similarities and several smaller variations [9].

4.4 Initial evaluation

Results	September 2002	October 2002
Number of special cases created	4	13
Average number of times used	1	5.4
Average time to create a special case	N/A	235 seconds
Number of times KANAL was run	32	42
Number of COAs tested	9	8
Total number of warnings/errors	85	345
Number of times user agreed	N/A	29
Number of times user disagreed	N/A	43

Table 3: Action Editor/KANAL usage

Between June and October of 2002, a number of tests were run in which two users entered COAs in a graphical tool, tested them with SHAKEN and KANAL and added knowledge to allow the tool to test the COAs including action special cases. The subjects were domain experts with no computer science background. After training, the only input given to the subjects was a textual description of two COAs. Although system developers were present to answer questions, they did not proactively give help and their interactions were controlled.

Table 3 shows the results from the two tests (in September 2002 and October 2002). During the two sessions the subjects entered seventeen new special cases to improve KANAL's performance on their COAs, taking an average of just under four minutes to enter each special case. KANAL was able to match almost all of the special cases to the COAs and behaved in the way the subjects had intended. The new cases were used more often (5.4 times) in the final test, resolving many of the errors and warnings found during the KANAL runs. When cases were not matched, this tended to be because the subjects had tried to use features that are not currently supported, such as negation. The subjects were generally positive to the approach, commenting for example that KANAL's behavior was 'surprisingly good' when special cases were applied.

During both in testing phase in September and in the final evaluation, the subjects entered special cases that we had not anticipated, modifying KANAL's behavior in novel ways. For example, the action model had a flaw that led to a certain unit strength being required to seize unoccupied terrain. The subjects created special cases that effectively corrected this flaw.

5. DISCUSSION

The current action editor demonstrates how a graphical approach can be used to allow users to modify the knowledge about actions used for plan checking in a broad plan authoring tool such as KANAL. There are several directions to more fully support users in managing action knowledge with this approach. At the moment, users can specify property and object information about the actions in order to change KANAL's behavior. We plan to support modifying preconditions and effects directly within the editor, either through a table or graphically. The same techniques will be used to enrich the information that can be used for the 'use-when' information that triggers an action special case.

The approach can also be extended to provide control information for both plan generation and plan checking, by authoring rules that determine when to choose a particular action in a plan. Extending KANAL to handle limited amounts of plan generation as well as the current plan checking would allow the user to specify goals or abstract steps at certain points in the plan for KANAL to flesh out. This would significantly increase KANAL's usefulness as an intelligent aid to a human planner, but to do this we need to expose more sophisticated information about actions than is currently done, to include information about the plan generation process and give the user some control over it. For example, the plan generator may need to know (1) plausible ways to generate values for action roles that are not specified in a goal or abstract step, (2) preferred ways to expand an abstract step and (3) preference information over alternative plans, among other things. The way that this kind of information is typically represented in AI planners would require users to have a detailed understanding of the planning algorithm in order to bring about a desired change in the planner's behavior. An interesting research goal will be to make this information accessible to end users through a combination of representation and user interface techniques in a similar fashion to the action special cases approach described here.

We are also investigating approaches for developing interactive acquisition dialogue where the system can provide help in deciding when, what, how and how well to build plans. This may be done by formulating acquisition goals and acquisition strategies over the current state of the plan that is being built. In developing acquisition goals and strategies, we are drawing ideas from tutorial dialogues and principles used in intelligent tutoring systems in order to make interactive acquisition interfaces as a proactive learner [14].

6. ACKNOWLEDGMENTS

We would like to thank Ken Murray for his help in building the graphical editor for entering special cases. Amit Agarwal and Varun Ratnakar have re-written the KANAL GUI code in order to provide a new look for KANAL. We thank Bruce Porter and Ken Barker at University of Texas at Austin who provided background knowledge needed for KANAL.

We would like to thank Yolanda Gil for her insightful comments on KANAL and the action editor. We also thank SRI team members including Vinay Chaudhri, Tomas Uribe, Peter Clark

and Sunil Mishra for their support. This research was funded by the DARPA Rapid Knowledge Formation (RKF) program with subcontract number 34-000-145 to SRI International under contract number N66001-00-C-8018.

7. REFERENCES

[1] Barker, K., Clark, P. and Porter, B., A Library of Generic Concepts for Composing Knowledge Bases. Proceedings of the First International Conference on Knowledge Capture (K-CAP-2001), pp. 14-21, 2001.

[2] Blythe, J., Kim, J., Ramachandran, S. and Gil, Y., An Integrated Environment for Knowledge Acquisition. Proceedings of International Conference on Intelligent User Interfaces (IUI-2001), pp.13-20, 2001.

[3] Blythe, J., Integrating Expectations to Support End Users to Acquire Procedural Knowledge. Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-2001), pp.943-952, 2001.

[4] Blythe, J.,SADL: Shaken Action Description Language, http://www.isi.edu/expect/rkf/sadl.html.

[5] Chien, S., Static and completion analysis for knowledge acquisition, validation and maintenance of planning knowledge bases. In International Journal of Human-Computer Studies, 48, pp. 499-519, 1998.

[6] Clark, P. and Porter, B., The knowledge machine. In http://www.cs.utexas.edu/users/mfkb/km.html.

[7] Clark, P., Thompson, J., Barker, K., Porter, B., Chaudhri, V., Rodriguez, A., Thomere, J., Mishra, S., Gil, Y., Hayes, P. and Reichherzer, T., Knowledge Entry as the Graphical Assembly of Components. Proceedings of the First International Conference on Knowledge Capture (K-Cap-2001), pp. 22-29, 2001.

[8] Cypher, A. Watch what I do: Programming by demonstration. Allen Cypher, Ed. MIT press, 1993.

[9] Desimone, R. and Agosta, J., Oil Spill Response Simulation: the Application of Artificial Intelligence Planning Technology, Simulation Multiconference, San Diego, 1994.

[10] Erol, K., Hendler, J., Nau, D., and Tsuneto, R., A Critical Look at Critics in HTN Planning. Proceedings of the 1995 International Joint Conference on Artificial Intelligence (IJCAI-95), pp. 1592-1598, 1995.

[11] Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D. and Wilkins, D., PDDL-the Planning Domain Definition Language, Technical report TR-98-003, Yale Center for Computational Vision and Control, October 1998.

[12] Kim, J. and Gil, Y., User Studies of an Interdependency-Based Interface for Acquiring Problem-Solving Knowledge. Proceedings of the Intelligent User Interface Conference (IUI-2000), 165-168, 2000.

[13] Kim, J. and Gil, Y., KANAL: Knowledge ANALysis on Process Models, In Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001), pp.935-942, 2001.

[14] Kim, J. and Gil, Y., Deriving Acquisition Principles from Tutoring Principles, Proceedings of the Intelligent Tutoring Systems Conference (ITS-2002), pp.661-670, 2002.

[15] Long, D. and Fox, M. "Automatic synthesis of design types in planning", In Proceedings of the Conference on Artificial Intelligence Planning Systems, 2000 (AIPS-2000), pp. 196-205, 2000.

[16] Mitchell, T., Mahadevan, S. and Steinberg, L., LEAP: A learning apprentice for VLSI design. Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI-85), pp. 574-580, 1985.

[17] Myers, K., Strategic advice for hierarchical planners. In proceedings of the international conference on knowledge representation and Reasoning, 1996 (KR-96), pp. 112-123, 1996.

[18] Pane, J., Ratanamahatana, C. and Myers, B. "Studying the language and structure in non-Programmer's solution to programming problems", International Journal of Human-Computer Studies, vol. 54, no. 2, February 2001, pp. 237-264, 2001.

[19] Simpson, R. McCluskey, T., Long, D. and Fox, M., "Generic types as design patterns for planning domain specification", 2002.

[20] Simpson, R., McCluskey T., Zhao W., Aylett, R. and Doniat, C., An Integrated Graphical Tool to support Knowledge Engineering in AI Planning. Technical Report - University of Huddersfield 2001.