Knowledge Analysis on Process Models

Jihie Kim and Yolanda Gil
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292, U.S.A.
jihie@isi.edu, gil@isi.edu

Abstract

Helping end users build and check process models is a challenge for many science and engineering fields. Many AI researchers have investigated useful ways of verifying and validating knowledge bases for ontologies and rules, but it is not easy to directly apply them to checking process models. Also the techniques developed for checking and refining planning knowledge tend to focus on automated plan generation rather than helping users author process information. In this paper, we propose a complementary approach which helps users author and check process models. Our system, called KANAL, relates pieces of information in process models among themselves and to the existing KB, analyzing how different pieces of input are put together to achieve some effect. It builds interdependency models from the analysis and uses them to find errors and propose fixes. Our initial evaluation focused on how useful KANAL is in helping users detect and fix errors. The results show that KANAL was able to find most of the errors and suggest useful fixes including the fixes that directly point to the sources of the errors.

1 Introduction

Building process models is essential in many science and engineering fields. Since some of these processes are quite complex, it is useful to provide tools to users that enable them to specify process models. Figure 1 shows an example of a process model in Cell Biology to describe how a Lambda virus invades a cell. To be able to specify such a model, the user has to specify each of the individual steps and connect them appropriately. There are different types of connections among the steps, including decomposition links between steps and substeps, ordering constraints, disjunctive alternatives, etc. Even in process models of small size, the number of steps and connections between them is large enough that users would benefit from the assistance of intelligent acquisition tools that help them specify process models correctly. For example, the user may either forget to specify the links between the steps, or may specify wrong links. We found such errors in a Biological Weapon production model built by a subject matter expert as a part of the DARPA Rapid Knowledge Formation

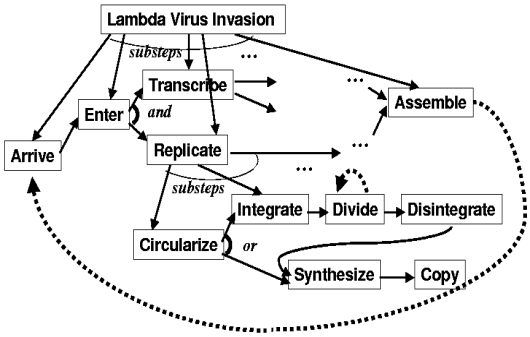

Figure 1: Example Process Model.

(RKF) program. Even though it can be considered as a relatively simple model (consisting of 54 steps) and many people had looked at it, there were at least two errors: (a) there were two steps that were both specified as the second substep of the same step; (b) there were other two sequential substeps whose ordering information was missing. Although these may be simple errors, in more complex models users may generate more serious problems and have difficulty noticing and fixing them.

Graphical tools to lay out a process model and draw connections among steps abound, but the tools are limited to simple checks on the process models because there is no semantics associated to the individual steps. In contrast, we assume a knowledge-rich environment that enables users to specify what the process and its steps mean, what they should accomplish and how, and what are the features of the objects involved in those steps. In order to check a process model such as the one in the figure throroughly, the system would have some background knowledge about what a cell is, that it can be entered because it is a physical object with a barrier, etc. It would also be helpful for the tool to know what a change of location is, and that the Enter step implies a change of location of its agent. Domain ontologies and upper or middle level ontologies are commonly used to represent this kind of background knowledge. With this context, the tool can be much more helpful in checking that the process model makes sense within the background knowledge that it has.

Past research in validation and verification of knowledge bases addresses the detection of errors in rule bases [Preece & Shinghal, 1994; O'Keefe & O'Leary, 1994] or in ontolo-

gies [McGuinness *et al.*, 2000] and has not addressed process models specifically. Research on interactive tools to acquire planning knowledge is also related [Chien, 1998; Myers, 1996; Huffman & Laird, 1995], but their focus is on acquiring knowledge about how to generate plans instead of acquiring the specific plans themselves. Plan authoring tools are closer in spirit to the process authoring tools that we are aiming for, and as such KANAL could be used to check errors in plans produced by plan editing tools. Much of the work on planning does not exploit background knowledge and ontologies, which we believe is crucial technology to advance the state of the art in process modeling.

We have developed a tool that checks process models specified by a user, reports possible errors in the models, and generates specific suggestions to the user about how to fix those errors. Our system is called KANAL (Knowledge ANALysis), and it helps users build or modify process models by detecting invalid statements and pointing out what additional knowledge needs to be acquired or what existing knowledge needs to be modified. Our approach is inspired on previous work on EXPECT using *Interdependency Models* (IM) [Swartout & Gil, 1995; Kim & Gil, 2000]. These models of the interdependencies between different pieces of knowledge can be derived by analyzing how knowledge is used during problem solving. By analyzing these interdependencies, a knowledge acquisition tool can detect inconsistencies and missing knowledge and alert the user of potential problems in the knowledge base. Finally, based on the context provided by the Interdependency Models it can guide the user in fixing these problems by correcting inconsistencies and by adding further knowledge. KANAL analyzes the interdependencies among the individual steps of a process model. For example, a simulation of the execution of the steps allows KANAL to analyze interdependencies between the conditions and effects of different 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.

The paper begins by describing the representation of process models assumed in KANAL. Then we describe how interdependency models can be applied to check various features of process models. Next, we present the current implementation of KANAL and the algorithms that it uses to detect different kinds of errors. Finally, we present the results from a preliminary evaluation that show that KANAL can detect most of the errors which are randomly introduced in originally error-free process models, suggesting useful fixes that often point directly to the source of the errors.

2 Representing Process Models

This section describes briefly our representation of process models, which is consistent with current efforts on standard languages and process ontologies, such as PDDL[Ghallab *et al.*, 1998] and NIST's PSL[Tissot & Gruninger, 1999].

A process model is composed of a number of (sub)steps. Each individual step has preconditions and effects, where the preconditions specify the conditions needed to be satisfied to activate the step and the effects describe changes that result from the execution of the step. For example, an "Enter" step has a precondition that the objects to enter should be near the entrance of a container object. Its effect can include a location change from outside of a space to inside of the space and also a status change to being contained within the container. These can be represented as precondition list and add/delete lists (as in STRIPS operators).

The steps within a process model are connected to other steps through different kinds of *links* including:

- decomposition links: Users can specify super-step/substep relations. For example, an Invade step can have Arrive, Enter and Take Control as its substeps, and each of these substeps can have their own substeps.
- temporal links: Users can specify ordering constraints among the steps. For example, in modeling virus invasion, the Take Control step should follow the Enter step.
- disjunctive links: There might be more than one way of performing a given task, and the alternatives can be represented by disjunctive links. For example, the DNA of a Lambda virus can either start its replication right after entering a cell or be integrated with the host chromosome for a while before the replication.
- causal links: Users can specify enablement/disablement between steps. Since KANAL can compute the actual causal relationships among the steps from the simulation results (by examining the outcome of the steps and the preconditions checked by other steps), the user-specified causal links can be used for validating the model.

Each step can have several *roles*. For example, in an Enter step an object can play the role of an agent and another object can play the role of the container being entered. A general description of an Enter step can be instantiated for the Virus invasion process by *assigning* the concept virus to the agent role of Enter and the concept cell to the container role. These role assignments cause further interdependencies in the knowledge base (KB), since the objects assigned to the roles have their own constraints and definitions that must be consistent with those of the process models and their steps.

3 Acquiring Process Models: The End-to-End System

Currently, KANAL is developed as a module within an ambitious end-to-end system that will support subject matter experts entering domain knowledge as a part of the DARPA Rapid Knowledge Formation (RKF) program. The user interface and the component library have not been fully implemented and integrated with KANAL, although a preliminary version of the three was done to illustrate the approach with a small scale scenario of a process model for virus invasion. In this project, users will build process models is by using "concept composition" [Clark & Porter, 1997]. Users can build process models by retrieving components (actions and objects) and then connecting them using various kinds of links.

Figure 2 is an interface that we built to show how the component approach can be used to build process models by link-

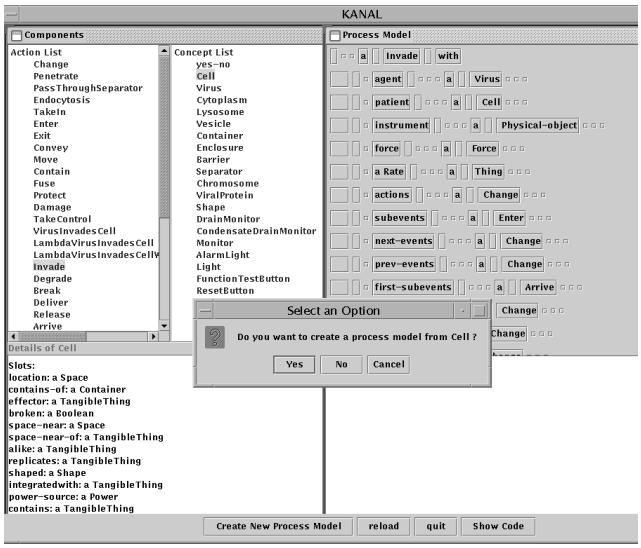

Figure 2: The KANAL interface for building process models.

ing various objects in the KB. The user has selected the Invade component to start building the Virus Invasion model. The "agent" role can be assigned to the Virus concept, the Enter component can be linked as a subevent, and so on.

Although KANAL is built to check process models constructed by concept composition, it can also be used for checking process models in other environments. For example, providing procedural knowledge required by intelligent tutoring systems has been an ongoing challenge, and they sometimes use simulators to build and refine their models [Scholer *et al.*, 2000]. We are also investigating the use of KANAL to help teachers formalize process models that will be used as lessons by the tutoring system, by checking errors and providing fixes until they finish building their models.

4 Using Interdependency Models

In past work, Interdependency Models have been used in analyzing how individual components of a knowledge base are related and interact when they are used during problem solving. An example of interdependency between two pieces of procedural knowledge is that one may be used by the other to achieve a subgoal. Other kinds of IM include interdependencies between factual knowledge and procedural knowledge. IM can point out missing pieces in solving a problem and be used to predict what pieces are related and how. They have been successfully used in building and checking problem-solving knowledge in EXPECT [Kim & Gil, 1999; Kim & Gil, 2000]. In this paper, we show a novel use of IM to check process models.

To guide users in developing process models, KANAL builds interdependencies among KB objects in the system, and uses them to perform two kinds of checks: static checks and dynamic checks. Static checks are performed by posing questions about various features of the process model, and dynamic checks are performed by simulating the execution of the process model. Our initial work to date has focused on dynamic checks.

In order to perform static checks, we plan to maintain a list of sample query templates, such as retrieving the values of different links, types of roles assigned to steps, etc. A list of instantiated queries can be generated for a particular model using these templates. Users could select key queries from this list and also specify the answers expected from the queries. A trace of the answer to a query can be considered as a model of the interdependencies in that it reflects how different pieces of knowledge are put together to generate the answer.

Dynamic checks can be done on the simulated execution of the process model. The simulation results show how different steps are related each other, including temporal ordering and causal relationships. The results also show how certain effects are produced by a set of sequences of steps. The resulting Interdependency Model enables checking if all the steps are properly linked, all the preconditions of each step are satisfied during the simulation, all the expected effects can be achieved, there are no unexpected effects, there are no impossible paths, etc. Also the interdependencies can point to potential ways of solving errors and gaps in the model, such as changing ordering constraints to reinstate disabled effects, finding steps that can generate unachieved effects, adding missing links, etc.

Our current work focuses on dynamic checks, using the simulation as a tool to generate Interdependency Models. The next section describes how we check the process models using simulation results.

5 Checking Process Models with KANAL

Our current implementation is built using the KM knowledge representation and reasoning system [Clark & Porter, 2000], and invokes its simulator to generate alternative simulations of a process model[1]. KM provides a function that can execute a step in a given situation and create a new situation based on the add/delete lists of the step. KANAL uses this function to execute the steps in the given model and check various kinds of problems. Whenever a precondition test fails or any of the events are *undoable* (a step is undoable when not all of its previous steps were executed.), KANAL interrupts the simulation and reports the *unreached* steps (a step is unreached when the simulation stops before the step is simulated). It also reports other problems found until it finishes the simulation. From the simulation results, including the problems detected during the simulation, KANAL can compute potential fixes through IM.

The subsections below describe each type of check that KANAL performs in detail.

5.1 Checking Unachieved Preconditions

A precondition is not achieved either because there is no previous step that produces the needed effect or because some previous steps undo the precondition. For example, an Integration of a virus DNA into a host chromosome may undo a

[1]KM's simulation of process models can be seen as a symbolic execution of a linearization of the process model using Skolem instances.

precondition (that the virus DNA is exposed) of a step to Synthesize protein from DNA. To be able to synthesize the viral protein needed for the replication, an additional Dis-Integrate step that can reinstate the exposure is required.

The general algorithm to check preconditions is as follows:

1. Notice problem
   (a) Run simulation with Skolem instances
   (b) Collect failed step(s)
   (c) Collect unachieved preconditions of failed step
   (d) Show them to user
2. Help user fix problem
   (a) *Suggest that there are missing steps*:
   - Find components in the KB that have the effects needed as preconditions by the failed step and suggest inserting one of these components somewhere within the current process model before the failed step
   (b) *Suggest that there are missing ordering constraints*:
   - Find steps that were executed before the failed step that may have effects that undid the unachieved preconditions
   - Find steps that follow the failed step and have effects that assert the unachieved precondition and suggest inserting an ordering constraint between those steps and the failed step
   (c) *Suggest modifying tthe step* whose preconditions were not achieved

For the above failure, KANAL suggests (a) adding a Dis-Integrate step, (b) changing or adding ordering constraints for the Integrate step, and (c) deleting or modifying the Synthesize step. Suggestion (a) would be the one that user is looking for in order to fix the problem in this example.

5.2 Checking Effects

The expected effects are what the user indicates should result from the simulation and/or the postconditions of the composed process model. For example, the user may expect that after a virus invasion of a cell, the virus should be located inside the cell. These expected effects can be checked by looking at the results from the simulation. The simulation results are represented as an accumulation of added and deleted facts. Since there may be multiple disjunctive branches in the model, the results from different paths are accumulated separately. KANAL checks the results from each path to see if all of them satisfied the expectation. If there are unachieved effects, KANAL can propose either to add new steps that would achieve them or to modify existing steps. Modifying existing steps can be either changing the values assigned to their roles or modifying the ordering constraints.

Currently KANAL checks for expected effects, but does not check explicitly for unexpected effects. We are planning to highlight any unexpected effects to let the users examine whether they should in fact occur.

The algorithm is as follows:

1. Ask user to specify expected effects
2. Notice problem
   (a) Run simulation with Skolem instances
   (b) Collect unachieved effects from each path and record the steps in the failed paths
   (c) Show them to user
3. Help user fix problem
   (a) *Suggest that there are missing steps:*
   Find components in the KB that have the effects needed and suggest inserting one of these components somewhere within the current process model
   (b) *Suggest modifying steps*:
   Find steps that may have effects that can potentially change the role values of the unachieved effects and suggest modifying those steps to achieve the effects needed
   (c) *Suggest that there are missing ordering constraints*:
   Find steps that may have effects that undid the expected effects and find actions that assert the expected effects and suggest inserting an ordering constraint in order to maintain the expected effect where needed

Following the example above, suppose that the user specifies that after the invasion occurs the virus should be located inside the cell. Suppose also that the user forgot to add the Enter step in the model of virus invasion. KANAL suggests (a) adding new steps, such as Move or Enter, that would change the location of the virus, (b) modifying Arrive since it is an existing steps that causes the virus to change location, or (c) changing/adding ordering constraints among these steps. The user would choose option (a) to add an Enter step, and the problem would be fixed.

5.3 Checking Unordered Steps

Sometimes the user may either forget to specify links between the steps, or may specify wrong links as in the Biological Weapon production example mentioned in the introduction. These problems may be detected by mapping the steps to the components in the knowledge base that have certain ordering constraints already specified for their substeps, or by running simulation and doing the checks as describe above for unachieved preconditions and effects. During the simulation, KANAL walks through the steps and substeps using the user specified decomposition links and ordering constraints. The simulation is interrupted if some steps cannot be reached because of lack of ordering constraints or the steps are undoable. KANAL highlights these problems and proposes changing or adding ordering constraints among the steps. (We do not show the detailed algorithm here because of lack of space.)

5.4 Checking Inappropriate Execution of Steps

KANAL can also find modeling errors by watching the execution of steps. For example, if some of the assertions to be deleted by a step are not true in the situation where the step is executed, these assertions are reported. Also, if a step produces no effect, i.e., it does not delete or add any assertions, then KANAL reports such problem as well. This type of problem can occur when the step's roles or attributes are assigned to the wrong objects during the composition. Also,

if its previous steps have incorrect assignments already and produced unexpected effects, then the following steps cannot be executed appropriately.

KANAL proposes modifying the steps by changing their role assignments or modifying previous steps.

5.5 Checking Invalid Expressions

During the simulation, KANAL checks the truth/falsity of many assertions, especially for the precondition tests and the expected effect tests. Whenever there are objects tested but undefined, KANAL reports the problem of accessing undefined objects (or invalid expressions).

5.6 Checking Loops

Loops are not necessarily a problem in process models. For example, the replication of DNA can be repeated multiple times. However, they can be unintended repetitions especially when the user defines loops across many steps. KANAL provides a warning for such cases to let the user check if the loops are in fact intended.

5.7 Checking Disjunctive Branches

When there are disjunctive branches in a model, the user may not notice that some of the combinations of alternatives should in fact not be possible. KANAL exposes different branches in the models by showing different alternative combinations of substeps. As in the case of loops, disjunctive branches are not necessarily a problem and KANAL simply informs the user about them.

5.8 Checking Causal Links

After the simulation, KANAL computes the interdependencies between the steps based on how some steps generated effects that satisfied the preconditions of some other steps. For example, synthesizing viral protein needed for replication enables the replication step. These causal links may not have been explicitly indicated by the user. KANAL informs users when it notices causal links in order to help them validate the model.

Notice that KANAL can detect the same error in multiple ways since one abnormality can lead to another. For example, missing an ordering constraint can make some steps unreached during the simulation, which can also lead to failed expected effects because of the unexecuted (unreached) steps. Since whenever there are unreached steps there tend to be failed expected effects, users may want to focus on fixing problems about the unreached steps first. The same case holds for failed preconditions and unreached steps because failed preconditions interrupt simulation, leading to unreached steps. To help avoid confusion, KANAL can selectively present fixes so that the user can concentrate on the actual source of the problem. For the case of failed preconditions and unreached steps, KANAL presents the fixes for the failed preconditions first, but lets the users check other fixes if they want.

Results w/o 1 link	Virus Invasion	Lambda Virus Invasion	Check Drain Monitor	total
# of test cases	19	28	9	56
# of errors	19	28	9	56
# of errors detected	18	28	9	55
total # of fixes	82	139	23	230
- # of direct fixes	17	31	8	56
Avg. # of fixes proposed	4.56	4.96	2.56	4.18 (avg)
# of errors with direct fixes	16	26	8	50

Results w/o 2 links	Virus Invasion	Lambda Virus Invasion	Check Drain Monitor	total
# of test cases	10	10	10	30
# of errors	20	20	20	60
# of errors detected	13	14	15	42
# of fixes proposed	76	28	23	127
- # of direct fixes	17	12	13	42
Avg. # of fixes proposed	5.85	2	1.53	3.02 (avg)
# of errors with direct fixes	13	13	13	39

Results w/o 3 links	Virus Invasion	Lambda Virus Invasion	Check Drain Monitor	total
# of test cases	10	10	10	30
# of errors	30	30	30	90
# of errors detected	13	16	18	47
# of fixes proposed	62	58	54	174
- # of direct fixes	14	16	15	45
Avg. # of fixes proposed	4.77	3.63	3	3.70 (avg)
# of errors with direct fixes	12	16	15	43

Table 1: KANAL checks for Process Models.

6 Preliminary Evaluation

KANAL is being integrated with concept composition, explanation tools and their interfaces in the end-to-end system mentioned above, and we are planning to perform an extensive user evaluation of this integrated system. The preliminary evaluation presented here focuses on how useful KANAL is in itself as a module to detect and fix errors.

To evaluate KANAL's help in detecting and fixing errors, we used three process models: a virus invasion process, a Lambda virus invasion, and a Check-Condensate-Drain-Motor procedure from a High Pressure Air Compressor (HPAC) domain which has been also used for acquiring process models (lessons) for intelligent tutoring systems [Scholer *et al.*, 2000]. The first and last ones were written by other researchers.

We evaluated how KANAL could help users with one important kind of error: if they forget to specify one, two, or three links or role assignments. For example, to specify the Virus Invasion model the user would need to make a total of 19 links and assignments if no errors are made. The test cases were generated by taking the original correct process models and randomly deleting a subset of the links or assignments that users would need to make.

The first rows in the tables show the number of cases tested and the second rows in the tables show the total number of errors in the test cases. KANAL was able to detect most of the errors when there is only one error (55 of 56). KANAL misses one case in Virus Invasion model because its Invade component has Container as its patient which should in fact be a Cell (a more special concept than Container), but there was no explicit violation in any of the checks KANAL performs. To be able to detect such problems we may need to examine slots tested in the model and check if they in fact belong to the concept. For example, Cells contain cytoplasm but not any Containers do in general.

KANAL missed some cases when more than one link were deleted (42 among 60 without 2 links, and 47 among 90 without 3 links). This is to be expected, since some errors interrupt the simulation, and other errors cannot be detected unless further steps are simulated, as described earlier.

The numbers of fixes (the fourth rows) shown in tables are based on the selected fixes described in the previous section. Among the fixes proposed by KANAL, there are *direct fixes* that directly point to the deleted links. Indirect fixes lead to these links but do not point to them directly. For example, when we delete the agent (virus) of the Move step in the Virus Invasion model, KANAL detects "inappropriate execution" of the step since its delete-list may contain assertions that do not exist (location of a Thing instead of location of the virus). In such case, KANAL proposes to either to modify the links of the Move step (direct fix) or to modify the steps before the Move step so that they can assert the location of the Thing (indirect fix). In many cases direct fixes should be more useful than indirect fixes. The number of direct fixes are shown in the fifth rows in the tables. For most of the errors detected, KANAL was able to provide at least one direct fix that can be used to fix the model.

The number of errors which had direct fixes were shown in the last rows in the tables. We show this because in some cases more than one fix may point to the same source. In the above case (a problem with the agent slot of the Move step), the precondition of the following Enter step (location of the virus should be near the cell) may also fail, and KANAL detects this "unachieved precondition". KANAL proposes to modify the Move step (the same fix as above) which changed the location of the Virus. The table shows that there were a few cases where KANAL detected errors but was not able to provide direct fixes. For example a step's role can refer to a deleted slot of another step, such as when the agent of the Enter step refers to the agent of the Invade step. If the Enter step failed because of the missing agent of the Invade step, then the failed step is different from the step with missing links, making it harder to find the sources of the problems. We are planning to examine how we can follow such links among the steps to trace back to the original sources. Some other cases of lack of direct fixes happened when there are multiple errors at the same time because some errors are hidden as described above. Also, KANAL's selective presentation of fixes helps, but we expect that fixing one problem at a time may be easier for end users.

In summary, our preliminary evaluations show:

- KANAL virtually always (115 of 116 test cases) detected an error and made suggestions to the user. The one case that KANAL missed was a process model that was perfectly consistent although it was overgeneral, which is a problem that can only be noticed by a user.
- Detecting an error impaired detecting others since posterior steps will not be executed in the simulation. KANAL detected 98%, 70%, and 52% of the errors in the case where one, two, and three links and assignments were missing, which means that it always detected at least one error in each of the process models that had more than one error. Once an error was fixed, the next error was always found by KANAL.
- For 91.6% of the errors detected, KANAL's fixes pointed directly to the source of the errors.

7 Discussion and Future Extensions

As we mentioned, we are planning to perform an extended evaluation with end users when KANAL is integrated with the end-to-end system described earlier. In doing so, we will also be able to test the usefulness of KANAL with different types of errors than the ones we show here, including selecting wrong components in the KB.

There have been a lot of verification and validation techniques developed in software engineering [Wallace *et al.*, 1996; Basili, 1987]. Although many of them are not directly applicable, there are many common issues in building process information, including efficiency, maintenance, cost, reuse, etc. We are planning to examine useful techniques developed for such issues.

KANAL is built for concept composition where components in the KB are assumed not to have any errors. However, in other environments, such as in acquiring procedural knowledge for intelligent tutoring systems, we cannot expect that the models of actions will always be correct. Often, incomplete operators are used to model procedures and they are refined based on instructor input or through autonomous learning by experimentation. We believe that KANAL will be also useful in such an environment. We are currently integrating KANAL and an intelligent tutoring system where an instructor (the user) can directly author models as well as demonstrate how things work.

Acknowledgements

We would like to thank Bruce Porter and Peter Clark for their help in integrating KANAL within the KM simulator, and Andrew Scholer for his help with the HPAC process model and with the simulator of the tutoring system mentioned. We would also like to thank Vinay Chaudhri, Mabry Tyson, Jerome Thomere, and other members of the DARPA RKF program for their comments and feedback on this work. 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.

References

[Basili, 1987] Basili, V. & Selby R. Comparing the effectiveness of software testing strategies. In *IEEE Transactions on Software Engineering*, 13(12), 1987.

[Chien, 1998] 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.

[Clark & Porter, 1997] Clark, P. & Porter, B. Building concept representations from reusable components. In *Proceedings of AAAI-97*, pp. 369-376, 1997.

[Clark & Porter, 2000] Clark, P. & Porter, B. The knowledge machine. In *http://www.cs.utexas.edu/users/mfkb/km.html*, 2000.

[Huffman & Laird, 1995] Huffman, S. & Laird, J. Flexibly instructable agents. In *Journal of Artificial Intelligence Research*, 3:271–324, 1995.

[Kim & Gil, 1999] Kim, J. & Gil, Y. Deriving expectations to guide knowledge base creation. In *Proceedings of AAAI-99*, pp. 235–241, 1999.

[Kim & Gil, 2000] Kim, J. & Gil, Y. Acquiring problem-solving knowledge from end users: Putting interdependency models to the test. In *Proceedings of AAAI-2000*, pp. 223–229, 2000.

[McGuinness *et al.*, 2000] McGuinness, D., Fikes, R., Rice, J. & Wilder, S. The Chimaera ontology environment In *Proceedings of AAAI-2000*, 2000

[Myers, 1996] Myers, K. Strategic advice for hierarchical planners. In *Proceedings of KR-96*, 1996.

[O'Keefe & O'Leary, 1994] O'Keefe, R. & O'Leary, D. Expert system verification and validation. In *Expert Systems with Applications: An International Journal*, 6(1): 57-66.

[Ghallab *et al.*, 1998] Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D. & Wilkins, D. Pddl - the planning domain definition language. Technical report, Yale University. Available at http://www.cs.yale.edu/pub/mcdermott/software/pddl.tar.gz.

[Tissot & Gruninger, 1999] Tissot, F., & Gruninger, M. NIST process specification language. Technical report, NIST.

[Preece & Shinghal, 1994] Preece, A. & Shinghal, R. Foundation and application of knowledge base verification. In *International Journal of Intelligent Systems*, 9(8): 683-702, 1994

[Scholer *et al.*, 2000] Scholer, A., Rickel, J., Angros, R. & Johnson, L. Learning domain knowledge for teaching procedural tasks. In *AAAI-2000 Fall symposium on Learning How to Do Things*, 2000

[Swartout & Gil, 1995] Swartout, W. & Gil, Y. EXPECT: Explicit representations for flexible acquisition. In *Proceedings of KAW-95*, 1995.

[Wallace *et al.*, 1996] Wallace, D., Ippolito, L. & Cuthill B. Reference information for the software verification and validation Process. In *NIST Special Publication 500-234*, 1996.