To appear in the Journal of Experimental and Theoretical Artificial Intelligence, 2009.

Principles for Interactive Acquisition and Validation of Workflows

Jihie Kim, Yolanda Gil, and Marc Spraragen

University of Southern California/
Information Sciences Institute
4676 Admiralty Way, Marina del Rey, CA 90292, USA
{jihie, gil, marcs}@isi.edu
phone:(310) 822-1511
fax: (310) 823-6714

Keywords: interactive knowledge acquisition, workflow creation, workflow validation, process models

Abstract

Workflows, also known as process models, are essential in many science and engineering fields. Workflows express compositions of individual steps or tasks that assembled together account for various aspects of an overall process. When workflows include dozens of components and many links among them, the creation of valid workflows becomes challenging since users have to track many interdependencies and constraints. This paper describes principles for assisting users to create valid workflows that are based on two knowledge acquisition systems that we have developed. A shared goal in these projects was to enable end users who do not have computer science backgrounds, such as biologists, military officers, or engineers, to create valid end-to-end process models or workflows. Our approach exploits knowledge-rich descriptions of the individual components and their constraints in order to validate the composition, and uses artificial intelligence planning techniques in order to systematically verify formal properties of valid workflows. Both systems analyze partial workflows created by the user, determine whether they are consistent with the background knowledge that the system has, notifies the user of issues to be resolved in the current workflow, and suggests to the user what actions could be taken to correct those issues.

1. Introduction

Workflows, also known as process models, are essential in many science and engineering fields [Malone et al., 2003; Taylor et al 2006; Tissot and Gruninger 1999]. Workflows express compositions of individual steps or tasks that assembled together account for various aspects of an overall process. Workflows provide a computational mechanism to manage the complexity of the interdependencies among tasks as well as their individual requirements. Workflows can be used to track and coordinate tasks in human organizations [Malone et al., 2003]. Our focus is on workflows that can be used to manage computations to carry out scientific simulations or complex data analyses. For example, workflows have been used to simulate seismic hazard [Maechlin et al., 2005; Gil et al., 2007; Deelman et al., 2006], analyze astronomy data [Deelman et al., 2005], and in a variety of biomedical applications [Voit 2000; Wroe et al., 2004; Kim et al., 2007]. In some cases, workflows are composed of distributed web services. In other cases workflows are composed of computations that can be submitted for execution in remote locations. Workflow systems can manage computations in distributed environments [Churches et al 2005; Altintas et al 2004] and with very large datasets [Deelman et al 05; Deelman et al 2006].

Because workflows can be very complex, a variety of workflow editors have been developed to enable users to assemble workflows graphically. These editors allow the user to browse through a library of components, select components for the workflow at hand, and link component inputs and outputs to express their interdependencies. However, these tools lack the assistance needed to support users in creating valid workflows. When workflows include dozens of components and many links among them, it becomes challenging for users to create valid workflows.

We have developed two knowledge acquisition tools to assist users in creating and validating workflows in two diverse application domains: CAT (Composition Analysis Tool) that assists users in composing computational workflows [Kim et al., 2004; Kim and Gil 2004], and KANAL (Knowledge ANALysis) that assists users in creating and validating process models in biology [Kim and Gil 2001]. A shared goal in these applications was to develop knowledge acquisition tools that enable end users who do not have computer science backgrounds, such as biologists, military officers, or building engineers, to create valid end-to-end process models or workflows. The two systems support different applications with different types of workflows and their underlying knowledge representations and supporting reasoning capabilities are diverse. In spite of these differences, these systems followed the same approach and incorporated similar principles:

1) exploit *knowledge-rich descriptions* of the individual components and their constraints in order to validate the composition

2) exploit *artificial intelligence planning* techniques in order to systematically verify formal properties of valid workflows

Both systems analyzed partial workflows created by the user, determined whether they were consistent with the background knowledge that the system had, notified the user of issues to be resolved in the current workflow, and suggested to the user what actions could be taken to correct those issues.

This paper presents techniques and principles that we have found to be central in helping end users author and validate workflows interactively. We begin with our approach in developing CAT (section 3) and then KANAL (section 4). In presenting each system, we start describing supporting knowledge bases that include two ontologies: ontology of data objects or entities (e.g. virus, rupture) and ontology of process components (e.g. move action, wave propagation). We show how these ontologies are used in reasoning about process models or workflows and about user actions involved in entering process models or workflows. We then present a set of desirable properties that we have developed based on existing planning techniques, and how we use them in validating process models and workflows against the ontologies. We discuss the underlying principles in both CAT and KANAL used to validate workflows and to assist users. Finally, we discuss our future work in developing additional support for interactive process model authoring and validation.

2. Motivation

Figure 1 shows a sketch of a workflow for seismic-hazard analysis. Scientists have developed many computational components (called models) that can be used to simulate various aspects of an earthquake: the rupture of a fault and the ground shaking that follows it, the shape of the wave as it propagates through different kinds of soil, the vibration effects on a building structure, etc. The models are complex, heterogeneous, and come with many constraints on their parameters and their use with other models. To create workflows composed of these models, users may follow different strategies and design the workflow in many different ways. One way is to think about it is in terms of simulation models. The users know they need two main steps: first, simulation of fault rupture; then, simulation of the wave propagation. They may prefer physics-based models or empirical models, and are a bit familiar with the scientific community and the methodology involved in creating each model. But another way to think about the workflow is in terms of the particular data the users want to look at. Sometimes they want the wave's velocity at the site, or its acceleration. Sometimes they want the probability of an earthquake above a certain magnitude affecting that site. Different models provide different types of results. Another way to think about

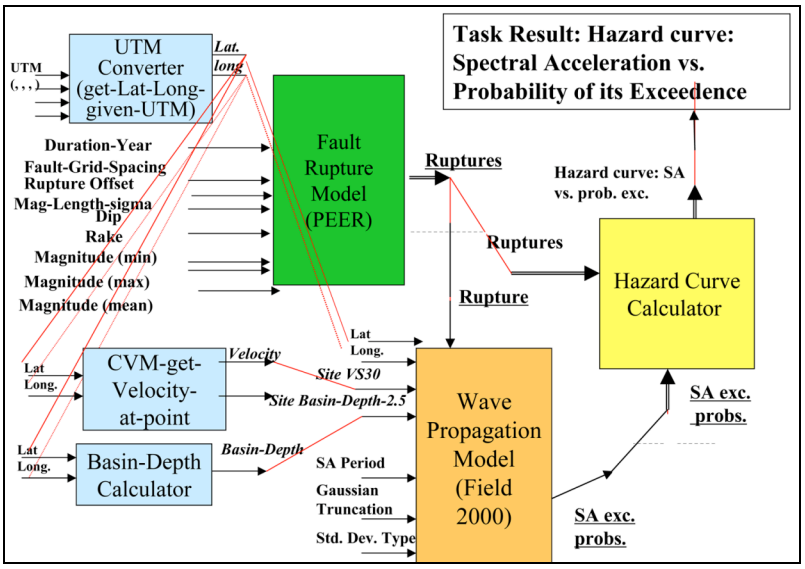

Figure 1. A computational workflow for earthquake simulation analysis.

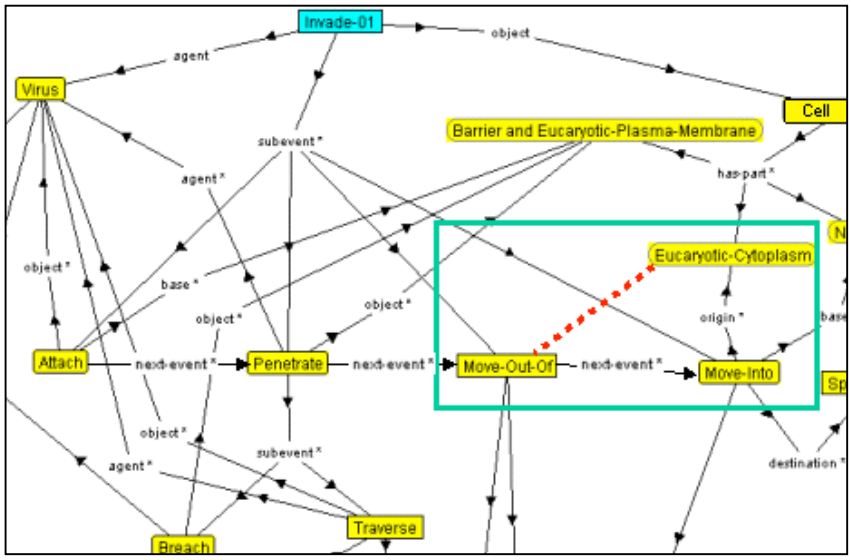

Figure 2: Example process model: a virus invading a cell.

the workflow is in terms of the situation the users want to simulate. The engineer may start with a specific site, then look at its characteristics like basin depth, and then select models that incorporate these characteristics. Generally, users need a system that is flexible enough to support the various strategies that they may take. In addition, in many cases end users have requirements and preferences that often depend on how the workflow unfolds and that cannot be specified beforehand. For example, users may not know

which wave propagation model is appropriate until the distance from the fault rupture to the location is determined.

Figure 2 shows a portion of a process model that was created by a biologist. It describes a general model of how a virus invades a eukaryotic cell. The model is composed of generic components that are predefined in a library, such as Move and Attach. There are different connections among the steps, including decomposition links between steps and sub-steps, and ordering constraints. Each step has several role assignments to the objects that play certain roles for the step, such as the Virus is the agent of the Penetrate step. This type of process model tends to have complex interactions among the steps; for instance, the Attach step (the Virus is attaching to the Eucaryotic-Plasma-Membrane) enables the following Penetrate step (the Virus is penetrating into the Cell). To be able to specify such a model, the user has to select each of the individual steps and connect them appropriately. Even in process models or workflows 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 or workflows correctly. For example, the user may forget to specify the links between the steps, or may specify wrong links. There are many constraints that users need to keep track of, including validity of the links and the steps added.

These examples shared requirements for supporting the creation and validation of workflows are:

- **keep track of details to ensure that a correct process model is formulated**: Authoring process models or workflows, as any user-driven process, is a task prone to errors and inconsistencies. As users edit the process model by adding components, linking their inputs and outputs, etc., there are many constraints that need to be tracked in terms of the validity of the links and the steps added.
- **support mixed-initiative interaction**: Users can drive the process when they have a clear idea of what to specify about the process model, whether they follow a top-down or a bottom-up approach, start from the desired outcome to initial steps or from available data, etc. At any point in time, the system should be able to take a partially specified process model or workflow from the user and make suggestions about how to complete it.
- **systematically generate and manage all of the choices throughout the authoring process**: At any point during the authoring, there may be many choices to make: add a step (and if so which one), add a link, replace an existing step with a more appropriate one, etc. Ideally, all these possible choices should be generated systematically, and they should be presented according to how each contributes to the configuration of the process model.

3. CAT: Interactive Composition of Workflows

This research was motivated by the need of scientists and engineer to create valid workflows for seismic hazard analysis, an application where using an invalid model or workflow can lead to dire consequences. Seismic hazard analysis enables building engineers to estimate the impact of potential earthquakes at a construction site and on their building designs. Scientists have developed many models that can be used to simulate various aspects of an earthquake: the rupture of a fault and the ground shaking that follows, the shape of the wave as it propagates through different kinds of soil, the vibration effects on a building structure, etc. Some of these models are based on physics; others are empirical based on historical data on past earthquakes. The models are complex, heterogeneous, and come with many constraints on their parameters and their use with other models. If a model is used in simulations where those constraints are violated, an erroneous seismic hazard would be estimated and therefore a construction may not be designed appropriately. Our goal is to enable unsophisticated users, such as building engineers and safety officials, to create end-to-end workflows composed of complex scientific models that are valid for the particular context in which they are used.

Whereas a workflow represents a flow of data products among executable components, a *workflow template* is an abstract specification of a workflow, with data types as placeholders for actual data products, and describing which components are used and how their parameters are connected. Our work focuses on workflow templates rather than workflows since users/scientists often create a workflow template and use and reuse the same template with different choices of data such as when they need to analyze potential seismic hazards on different sets of sites. A workflow that can be executed is created by binding actual data to data types in a workflow template. The components that are introduced are represented as *workflow template steps* which we simply call *steps* whenever the interpretation is clear by context. In a workflow template, an output parameter of a step can be linked to an input parameter of another step, so that data can pass between the steps. A *link* is a matched pair, consisting of one output parameter and one input parameter.

A workflow template includes *initial input* data types, which are linked to some steps' input parameters. All initial input data is assumed to be available before the workflow's computation starts. A workflow template should contain at least one *end result* data type, which is a placeholder for the actual data that the workflow is meant to produce. In CAT, input data and end results are handled uniformly as any other steps, the former as steps with no input parameters (initial-input steps) and the latter as steps with no outputs (end-result steps). That is, each user-given input object type is represented as the output parameter of an initial-input and each end result type is represented as an input parameter of an end-result.

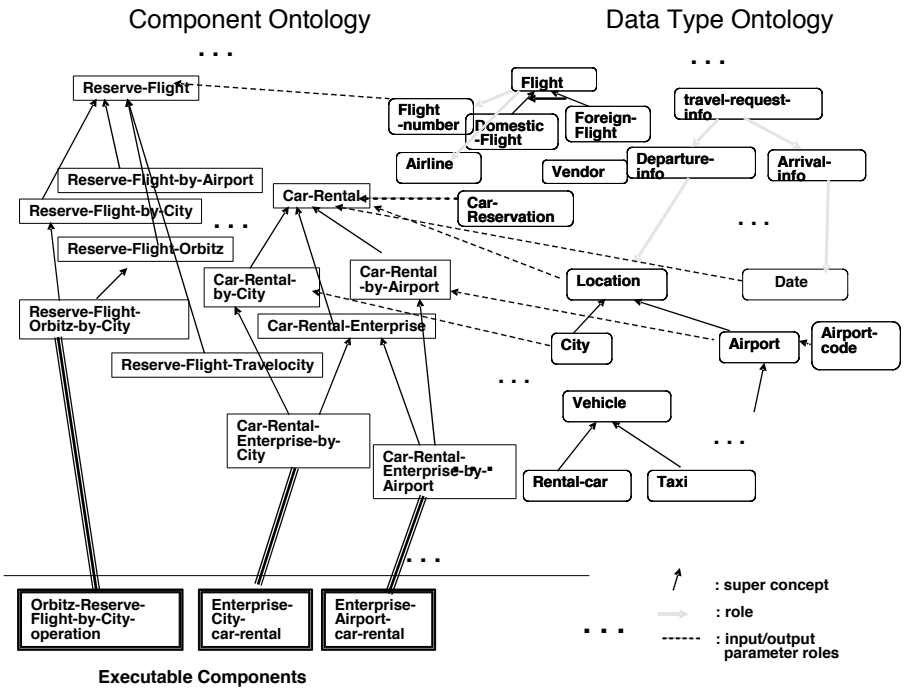

Figure 3. An example CAT Knowledge Base.

3.1 Supporting knowledge base: representing entities and workflow components

This section describes how we represent workflow components and data. Figure 3 shows a portion of a sample CAT Knowledge Base (KB). Here we use an example from a travel planning domain for readability. In the KB, there is a hierarchy of data types.

A *workflow component* represents an executable software program and is represented in terms of its input and output *parameters*, each with a data type expressed in the KB ontology. For example, component Car-Rental-at-Airport has a corresponding executable program. The component takes an Airport (as arrival-place) and a Date (as arrival-date) as input, and produces a Car-Reservation. We assume that the outputs of components produce new information and there are no deletions of existing information. *Abstract components* may have more abstract types of parameters. For example, a more abstract component Car-Rental has an input parameter arrival-place with type Location instead of Airport.

CAT uses OWL (the W3C Ontology Web Language standard) [OWL, 2008] for representing data types, but can access different KBs using the following generic KB access functions:

- **KB-components()**: returns a set of available components (including abstract ones) defined in the KB.
- **KB-data-types()**: returns a set of data types defined in the KB.
- **KB-input-parameters(c)**: returns input parameters of component c.

- **KB-output-parameters(c)**: returns output parameters of component c.
- **KB-executable(c)**: returns false iff c is not an executable component.
- **KB-range (c, p)**: returns a class defined (or derived) as the range of parameter p of component c. Here we assume that there is only one class that represents the range of the given class component and parameter.
- **KB-subsumes (t1, t2)**: returns true iff KB class t1 subsumes KB class t2.
- **KB-specializations(c,r,v)**: returns subconcepts of c, optionally where value for role r is v.
- **KB-component-with-output-data-type(t)**: returns a set of components c ∈ KB-components() s.t. ∃ p ∈ KB-output-parameter(c) ∧ KB-subsumes(t, KB-range(c,p)).
- **KB-component-with-input-data-type(t)**: returns a set of components c ∈ KB-components() s.t. ∃ p ∈ KB-input-parameters(c) ∧ KB-subsumes(KB-range(c,p),t).

For example, for the abstract component Car-Rental:

KB-input-parameters (Car-Rental) = {arrival-date, arrival-place}

KB-range (Car-Rental, arrival-date) = Date

Using the KB, CAT can reason about the semantics of each parameter and component to help users construct correctly formulated dataflow links in their workflows. Because the component ontology describes abstract component types as well as specific components, users can start from a high-level description of what they want without knowing the details of what actual components are available. We often find that users have only partial description of what they want initially, and our tool can help users find appropriate ones by starting with a high-level component type and then specializing it. The ontology of data types can be used in a similar way when users have incomplete or high-level description of the desired outcome. These ontologies also play a key role in relating components in workflows, detecting gaps and errors, and producing suggestions. For example, a link between an output of a component to an input of another can be checked to see whether the output type is subsumed by the input data type. The hierarchy of component types can guide the user to specialize an abstract-level component into one he/she likes. The next section describes how the KB contents are used within an algorithm that validates workflows and assists users to fix any errors or gaps.

3.2 Composition of workflows

The analysis of partial workflows created by the user is done using an AI planning framework [Nau 2007]. Formal analyses of planning algorithms define some desirable properties of plans, such as justifiability and correctness [Tate, 1996, Kambhampati et al., 1995, Yang, 1990]. Generative planners

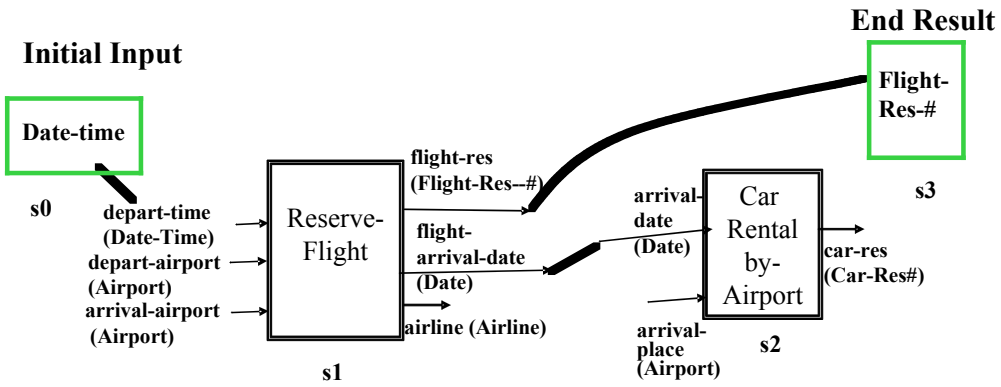

Figure 4. A simple workflow template.

[Wilkins, 1988; Sacerdoti, 1977; Tate, 1977] use critics that detect problems in the plans that they generate while planning. Some planning approaches have been used in composing workflows automatically [Blythe et al., 2003, McDermott 2002, McIlraith and Son 2001, Lansky et al., 1995, Chien et al., 1996]. While automating the creation of the workflows is not desirable, as users prefer to specify what workflow template must be used for the analysis, automatic planning approaches are not always appropriate. However, we use these general properties of plans to formally describe valid workflows.

Each workflow component is treated as a step in the plan, the inputs of the component are the preconditions of that step and the outputs are its effects, the links between steps are treated as causal links, any data provided by the user form the initial state, and the desired end results are the goals for the planning problem. There are no negative effects (deletions) as described above. Links in a workflow represent dataflow between input and output parameters of the steps.

Formally, a workflow template wt is a tuple <*S, L, I, G*> where *S* is a set of steps, *L* is a set of links, *I* is a set of initial-input steps and *G* is a set of end-result steps. Each *step* is a tuple <c, id> where c is a component in the KB and id is an identifier for the step. Each *link* is a tuple $<s_o,p_o,s_i,p_i>$ where p_o is an output parameter of a step $s_o \in S$, and p_i is an input parameter of $s_i \in S$. In our current framework, a workflow is a directed acyclic graph: components are nodes and links are directed edges. For example, the link between flight-arrival-date output parameter of Reserve-Flight step and arrival-date input parameter of Car-Rental-by-Airport step in Figure 4 can be represented as <<Reserve-Flight,s1>, flight-arrival-date, <Car-Rental-by-Airport,s2>, arrival-date>. For a step that is not an initial input or an end result, the input and output parameters of the step are the input and output parameters of the component. For example, input-parameters(<c, s_id>) = KB-input-parameters(c). Also, range (<c, s_id>, p) = KB-range(c,p).

User Activities for Workflow Creation

Figure 5 shows the web-based interface of CAT. On the left side are components and links currently in the workflow. The right side shows current errors or gaps in the workflow, as well as suggestions for the selected error.

The composition process is user-driven; at any time, the user may perform one of four primitive actions:

- **AddStep(wt,s)***:* Given a workflow template wt = <*S, L, I, G*> and c ∈*components*(), w becomes < *S* ∪{<c, s_id>}*, L, I, G*>. where s_id is a unique step identifier generated by the system.
- **RemoveStep(wt,s)***:* Given a workflow template wt=<*S, L, I, G*> and s ∈ *S*, wt becomes < *S* - {s}, *L, I, G*>.
- **AddLink(wt,s1,p1,s2,p2)***:* Given a workflow template wt=<*S, L, I, G*>, s1 ∈ *S,* p1∈ output-parameters (s1), s2 ∈ *S,* and p2∈ input-parameters (s2), wt becomes <*S, L* ∪ {<s1,p1,s2,p2>}, *I, G*>.
- **RemoveLink(wt, l)**: Given a workflow template wt=<*S, L, I, G*>, l ∈ *L,* wt becomes < *S, L- {l}, I, G*>.

Adding or removing initial-input and end-result steps is handled similarly to adding or removing regular steps. Each action taken by the user (add/remove component, add/remove link) is akin to a refinement operator in plan generation. However, while automatic systems can explore the space of plans systematically and guarantee that the final plans are valid, interactive composition requires an approach that lets the user decide what parts of the space to explore and that can handle incorrect partial workflow templates.

In addition to these primitive actions, CAT makes use of "composite" actions in order to make the composition process more coherent and efficient. Each composite action is an ordered sequence of the four primitive actions. Currently, we have four composite actions: AddAndLinkStep, RemoveStepAndLinks, SpecializeStep (remove existing step and add a new one with a more specialized component), and InterposeStep (remove an existing link, add a step, and add two links between the output parameter and input parameter of the removed link).

3.3 Desirable properties of a workflow template

This section first introduces some features of workflow template steps and links, and defines desirable properties that CAT uses in verifying user-entered templates.

- **Purposeful**: A workflow template should contain at least one end-result step.

  $\forall$ wt $<S, L, I, G>$, *purposeful* (wt) iff $\exists\ G \neq \phi$. That is. Even though CAT allows users to construct sketches of workflows without specifying desired end results, to complete a workflow, users need to provide the kinds of outcome they expect.
- **Grounded**: To be able to execute a given workflow template all the steps introduced to the workflow template should be specialized into executable ones.

  $\forall$ wt $<S, L, I, G>$, *grounded* (wt) iff $\forall$ s $\in S$, executable(s)
- **Satisfied**: All the inputs should be provided.

  $\forall$ wt $<S, L, I, G>$, *satisfied* (wt) iff $\forall$ s $\in S \cup G$, satisfied(s).
- **Consistent**: All the input/output links are valid in terms of the data types used.

  $\forall$ wt $<S, L, I, G>$, *consistent* (wt) iff $\forall$ link l $\in$ L, consistent (l).
- **Justified**: All the steps are used in achieving some goals.

  $\forall$ wt $<S, L, I, G>$, *justified* (wt) iff $\forall$ s $\in S$, justified (s).
- **Well-Formed**: All the steps and links are valid.

  $\forall$ wt $<S, L, I, G>$, *well-formed* (wt) iff $\forall$ s $\in S$, valid (s) $\wedge$ $\forall$ l $\in L$, valid (l).
- **Acyclic**: No cycles are included.

  $\forall$ wt $<S, L, I, G>$, *acyclic* (wt) iff $\neg(\exists$ s $\in S$, connected(s, s)).
- **Parsimonious**: Redundant links should be avoided.

  $\forall$ wt $<S, L, I, G>$, *parsimonious* (wt) iff $\neg(\exists$ l $\in L$, redundant (l)).

Table 1: Desirable Properties of Workflow Templates.

We define the following set of properties of steps and links:

- **Satisfied parameter and statisfied step**: wt = $<S, L, I, G>$, $s_i \in S$, $\forall$p $\in$ input-parameters (s_i), *satisfied* (p) iff $\exists$ a link $<s_o,p_o,s_i,p_i> \in L$ s.t. p_i = p. That is, an input parameter is satisfied when it is linked to any output parameter of a step. A step is satisfied if all its input parameters are satisfied.
- **Consistent link**: wt $<S, L, I, G>$, l $<s_o,p_o,s_i,p_i> \in L$, *consistent* (l) iff KB-subsumes (range(s_i,p_i), range(s_o,p_o)).
- **Valid step**: wt $<S, L, I, G>$, <c, s_id> $\in S$, *valid*(s) iff c $\in$ KB-components().
- **Valid link**: wt $<S, L, I, G>$, l $<s_o,p_o,s_i,p_i> \in L$, *valid*(l) iff $p_o \in$ output-parameters (s_o) $\wedge$ $p_i \in$ input-parameters (s_i).
- **Executable step**: wt $<S, L, I, G>$, <c, s_id> $\in S$, *executable*(s) iff KB-executable(c).
- **Connected steps**: $\forall$ wt $<S, L, I, G>$, $s_1 \in S \cup I$, c_1 , $s_2 \in S \cup G$, *connected* (s_1 , s_2) iff ($\exists$ link l $<s_o,p_o,s_i, p_i> \in L$ where $s_o = s_1 \wedge s_i = s_2$) or ($\exists$ step $s_3 \in S$ s.t. connected(s_1, s_3) $\wedge$ connected(s_3, s_2)). That is, there exists a (directional) chain of links that connects s_1 to s_2 in the workflow template.
- **Redundant link**: $\forall$ wt $<S, L, I, G>$, l $<s_o,p_o,s_i,p_i> \in L$, *redundant*(l) iff ($\exists$ link l_2 $<s_o',p_o',s_i', p_i'> \in L$ s.t. l $\neq l_2 \wedge s_i = s_i'$ and $p_i = p_i'$). That is, more than one link leads to the same input parameter.

- **Justified step**: ∀ wt <*S, L, I, G*>, s ∈ *S, justified*(s) iff s ∈ *G* ∨ ∃ s_2 ∈ *G* s.t. connected(s , s_2). Otherwise, S is *unjustified*. Currently, the Car-Rental-by-Airport step in Figure 4 is not justified.

For example, in Figure 4, the steps <Reserve-Flight,s1> and <Car-Rental-by-Airport, s2> are not satisfied yet. The initial-input step with output parameter Date-Time is *connected* to the end-result Flight-Res-# via the Reserve-Flight step.

Table 1 lists a set of desirable properties of workflow templates based on these properties of steps and links. When all the desirable properties are satisfied, the workflow template is considered *correct*. Although CAT relies on static analysis of workflow templates instead of dynamic simulation, and actual checks made are different, we can map some of these properties to relevant KANAL checks, as described below.

The properties can help relate the workflow templates generated by a user to templates that an automated approach could generate. Workflows that contain errors of inconsistency and redundancy would never be generated automatically. Automated approaches would normally form (partial) workflow templates that are well-formed, purposeful, justified, consistent and parsimonious.

3.4 ErrorScan algorithm

Based on the properties defined above, we have developed the ErrorScan algorithm. ErrorScan relies on a static analysis of 'scanning' the composed workflow template instead of results from dynamic simulations. Using the desirable properties, it produces a report on the kinds of problems that made the workflow template not correct. Each deviation from these properties, by the workflow or by one of its elements is reported as an error or a warning to the user. Based on the analysis of user actions shown above, the algorithm also generates specific suggestions to the user for how to fix each error found. Also, any fix suggested by CAT is an ordered sequence of the primitive actions above. The algorithm is shown in Table 2. The algorithm consults the knowledge base to check the properties (e.g., the consistency of a link based on the parameter type definitions in the ontologies), and to generate suggestions (e.g., if an input parameter is not satisfied, ErrorScan will return from the knowledge base a list of components that have outputs that are subsumed by the input parameter). If ErrorScan does not generate any errors or warnings for a given workflow template, the workflow is purposeful, grounded, satisfied, consistent, justified, well-formed, acyclic, parsimonious, and therefore it is a correct workflow template. If the workflow template is not justified, not parsimonious, or cyclic (i.e., capable of producing an end result but possibly inefficient), the system produces warnings instead of error messages. Many scientific workflows form directed acyclic graphs and we provide warnings in case a cycle is unintended.

ErrorScan

Input: a (partial) workflow template wt <*S, L, I, G*>
Output: list of errors and corresponding fix suggestions
I. **If wt is not purposeful, create an Error.**
Suggestions: define an end-result e using data types defined in the KB.
II. **If wt is cyclic, create Warning.**
Suggestions: for some links l in the cycle, **RemoveLink** (l).
III. For each step s in wt:
a. **If s is not justified, create a Warning.**
Suggestions: ∀p ∈ *output-parameters* (s), for a step sj in wt s.t.
pj ∈ *input-parameters*(sj), ∧ KB-*subsumes*(pj,p), **AddLink** (s,p,sj,pj).
b**. If the step <c, s_id> is not executable, create an Error.**
Suggestions: ∀ ci ∈ KB-*specializations*(c), **SpecializeStep**(s, ci).
c. For each pi ∈ *input-parameters*(s):
If pi is not satisfied, create an Error.
Suggestions: ∀ sj ∈ *S* with output parameter pj s.t. *KB-subsumes*(*range*(s, pi),*range*(sj,pj))
AddLink(sj,pj,s, pi).
Suggestions: ∀ cj ∈ KB-*component-with-output-data- type*(pi) with pj where
KB-*subsumes*(*range*(s,pi),KB-*range*(cj,pj)), **AddAndLinkStep**(cj,pj,s,pi).
Suggestions: add initial-input for pi with dj ∈*KB-data-types*(), s.t.
KB-*subsumes*(*range*(s,pi), dj).
d**. If the step s is not valid, create an Error.**
Suggestions: delete s.
IV. For each link l <so,po,si,pi> in w:
a. **If l is not consistent,**
If ∃ cj ∈ *KB-component-with-input-data-type*(*range*(so,po)) ∩
KB-*component-with-output-data-type*(*range*(si,pi)), **create a Warning.**
Suggestions: ∀ cj ∈ *KB-component-with-input-data-type*(*range*(so,po)) ∩
KB-*component-with-output-data-type*(*range*(si,pi)), **InterposeStep** (ci, l).
Otherwise, create an Error.
Suggestion: **RemoveLink(**l).
b. **If l is Redundant, create a Warning.**
Suggestion: **RemoveLink** (l).
c**. If l is not valid, create an Error.**
Suggestion: **RemoveLink** (l).

Table 2: The ErrorScan algorithm: Verifying user entered workflow templates

The algorithm filters its choice of suggestions, in that each suggestion must be a sequence of actions that, as a whole, fixes more errors than it causes. The suggestions are additive or corrective, i.e., the system will not suggest removing a valid component, though the user can always remove components on their own accord if they were added in error. The algorithm also incorporates heuristics for ordering errors as they appear in the interface. Errors are ordered most recent first (i.e., generated by the most recent user actions). Errors are then ordered by the more serious errors before warnings.

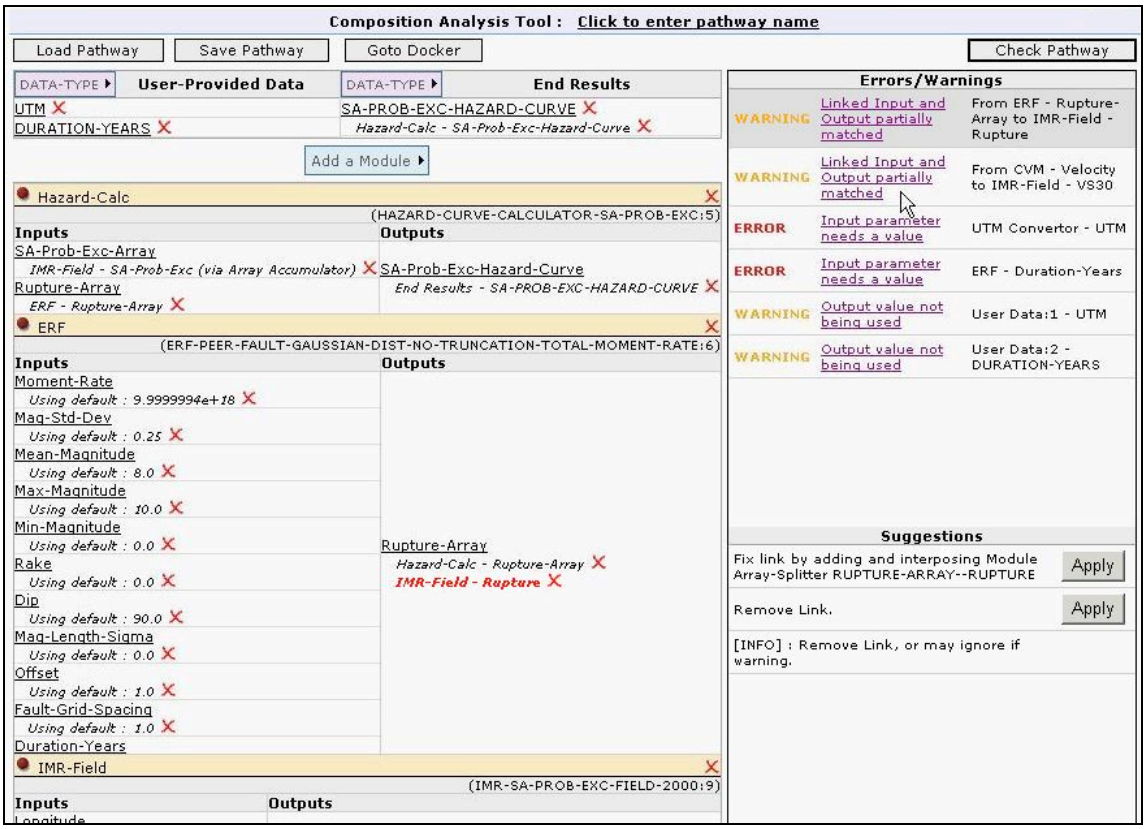

Figure 5. CAT User Interface, showing on the left side a few components of the workflow and on the right side the errors and suggestions found automatically by the system.

3.5 Example validations made by CAT

As the user creates a template, intermediate stages result in incorrect templates. Figure 5 shows example errors or defects in a workflow template, and how CAT detects them and proposes suggestions. Currently there are a couple of missing inputs for workflow components: UTM for UTM converter and Duration-Year for ERF. There are two warnings on two output values not used in achieving desired end results. The system also notices two inconsistent links that can be fixed by interposing a component: a link that connects Rupture Array of ERF and Rupture of IMR and a link from Velocity of CVM to VS30 of IMR-Field. CAT provides suggestions for the selected errors that interposing Array-Splitter can fix the inconsistent link problem. Currently, multi-step fixes are also available where the user can click-through the presented fix steps to complete the fix. If there is no error or warning detected by CAT, the template is *correct* and can be used for workflow creation (i.e. assigning data products to input parameters) and execution [Gil et al., 2006].

Our evaluations with synthetic user mistakes and workflow defects that are generated randomly show that CAT detects most of the defects (238 out of 240 defects). CAT missed the two cases due to the

interaction between the defects. For example, when a link is removed and then a component that contained the link's input parameter is also removed (i.e., the link would no longer need to supply a value), CAT may not detect the link had been removed. For 87% of the defects detected, CAT generates direct fixes (i.e. fixes that directly undo the defects generated) [Kim et al., 2005]. Besides the interactions between the defects, when abstract components are introduced (by replacing existing ones with more abstract ones), currently CAT proposes to specialize the components using only their children and may not point to direct modifications to the correct components. However, sequences of suggestions to specialize abstract components will eventually lead to desired components.

CAT has been integrated into an end-to-end SCEC (Southern California Earthquake Science) grid workflow execution system where users can create workflow templates, specify data files using metadata catalog service, refine the workflows to executable forms and use a grid environment to execute workflows against domain specific software libraries and data sources [Maechling et al., 2005; Gil et al 2007].

4. KANAL: Interactive Composition of Process Models

KANAL was built to help biologists enter complex process models in cell biology. KANAL was later extended to support critiquing of manually built courses of action that are process models of activities to be undertaken by a military unit. KANAL was developed within an end-to-end knowledge acquisition system called SHAKEN [Clark et al., 2001]. The resulting knowledge was used in answering cell biology textbook questions. Users can invoke KANAL whenever they want to validate their definitions of process models.

KANAL helps users build or modify process models by detecting possible errors and pointing out what additional knowledge needs to be acquired and what existing knowledge needs to be modified [Kim and Gil, 2001]. These operations use 1) knowledge-based descriptions of components that are used in process models, which support reasoning about inter-dependencies between steps and links that are introduced and 2) assessment of partially constructed process models based on AI planning and process modeling techniques.

4.1 Supporting knowledge base: representing data entities and actions

This section introduces the supporting knowledge base and representations that we use for process models in biology and military process models. The current implementation of KANAL is built on the KM knowledge representation and reasoning system. KM provides frame-based language with

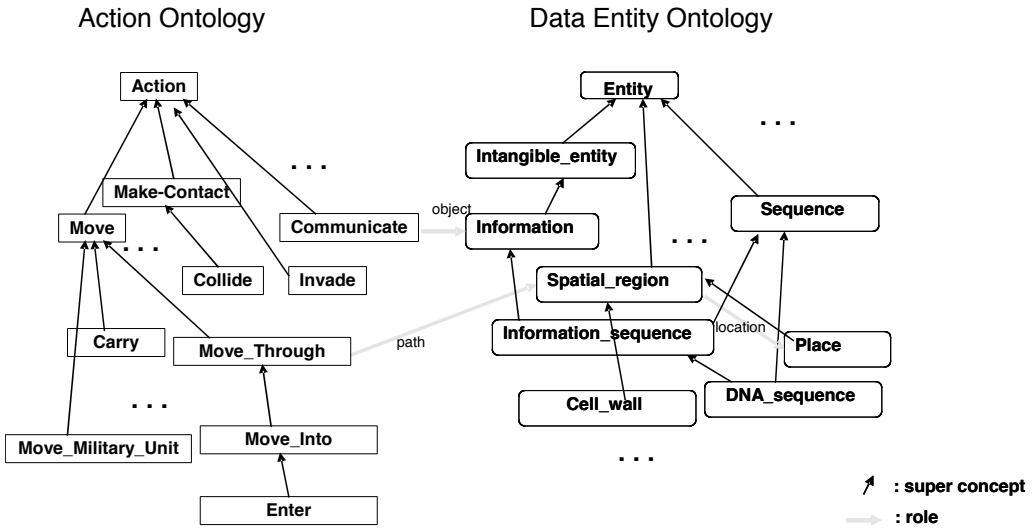

Figure 6. A portion of KANAL Knowledge Base : Only some parts are shown for brevity.

first-order semantics [Clark and Porter, 2005]. The Component Library defined in KM consists of domain independent upper ontology with about 80 semantic relations and about 500 generic concepts of *entities* and *actions* [Barker et al., 2001]. Each entity or action definition is a *concept* and consists of a small set of first-order logic axioms. Entity definitions specify objects, object aggregates, sequences, spatial entities, etc. that are involved in defining actions. In the biology domain, they include taxonomy of viruses, cells, nucleuses, etc.

An action can have *preconditions*, *effects*, and *roles*. 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" action has a precondition that the entities or 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 a precondition list and add/delete lists as in STRIPS operators [Fikes and Nilsson 1971]. Each action can have several *roles*. For example, in an Enter action an object can play the role of an agent and it is a tangible entity. Another object can play the role of the container being entered. Roles of an action can have their own constraints, such as: the destination of an Enter action should be inside the container. Some actions define how they are decomposed into several substeps. Figure 6 shows a portion of the knowledge base.

The following shows some of the supporting KB queries that KANAL makes in validating process models.

- KB-concepts(): returns a set of concepts (objects and existing process models) defined in the KB.
- KB-preconditions(c) : returns preconditions of an action c.
- KB-effects(c) : returns effects of an action c.
- KB-required-roles (c): returns required roles for a concept c.
- KB-range (c, r): returns a concept defined (or derived) as the range of role r of c.
- KB-subsumes (c1, c2): returns true iff c1 subsumes c2 in the KB.
- KB-specializations(c): returns subconcepts of c.
- KB-components-with-effect (e): returns a set of components that have an effect e.
- KB-potential-effect-of-step (c,e): returns a set of role assignments for action c that can produce an effect e.

4.2 Composition of process models

The representation of process models is consistent with standard AI planning languages and process ontologies, such as PDDL [Ghallab et al., 1998] and NIST's PSL [Tissot and Gruninger, 1999]. In presenting the axioms about an action to the user, the raw axioms are not presented directly. Rather, the user sees an example of that action, i.e., a set of ground facts about computed from those axioms. A process model is composed of a number of steps and substeps. Each individual step is an instance (example) of an action defined in the knowledge base (e.g. Enter).

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

- decomposition links: Users can specify superstep/ substep relations. For example, an Invade step can have Attach, Penetrate 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 a virus invasion, the Penetrate step should follow the Attach step.
- disjunctive links: There might be alternative actions for a process step, 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 before the replication.
- causal links: If the editor allows, users may specify enablement/ disablement between steps, and KANAL can compute the causal relationships among the steps inferred from simulation results, as described below. 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.

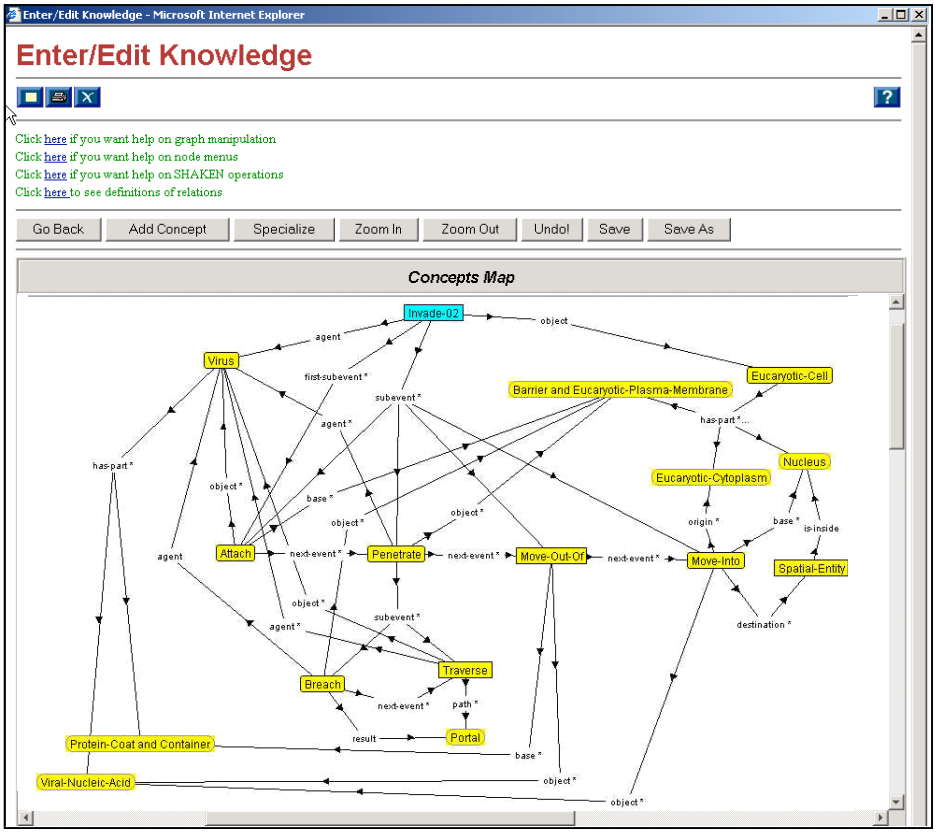

Figure 7: Process model editor [Clark et al., 2001]

User Activities for Process Model Creation

A general description of a Penetrate step can be instantiated for the Virus invasion process by *assigning* the concept Virus to the agent role of Penetrate and the concept Eucaryotic-Plasma-Membrane to the object role. These role assignments cause further interdependencies in the 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.

The following are the (user) actions for refining process models:

- add a step to the process model using an action definition
- remove an existing step
- add/remove a link between two steps
- add/remove a role assignment

Figure 7 shows the graphical editor that allows the user to specify a process model in SHAKEN. The design of the interface was inspired in concept maps [Novak 1998].4.3 Desirable properties of process models.

As with CAT, we turn to AI planning systems to derive properties that KANAL uses to validate process models. The properties are adapted to the process model representation and the composition steps used in KANAL. The properties are:

- All the preconditions of a step should be achieved before the step is executed.
- All the expected effects should be achieved.
- All the steps should be ordered.
- All the steps should be executable and they produce some effect.
- All the expressions should be valid.
- All the loops should be intended by the user.
- All the links (decomposition, temporal, disjunctive) should be intended by the user.
- All the causal relations should be intended by the user.

For loops, links, and causal relations, the properties state that the user should have intended those assertions. The purpose of those properties is to avoid that such assertions are created unintentionally as a side-effect of something else the user does during the creation of the process model.

4.4 KANAL Algorithm for checking process models

KANAL invokes KM's simulator to generate alternative simulations of a process model. KM's simulation of process models can be seen as a symbolic execution of a linearization of the process model using Skolem instances. 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. Table 3 summarizes the checks that are made by KANAL and suggestions provided by the system. They enforce the desirable properties of process models that are described above.

Each action taken by the user is akin to a refinement operator in plan generation. However, while automatic systems can explore the space of plans systematically and guarantee that the final plans are valid, interactive composition requires an approach that lets the user decide what parts of the space to explore and that can handle invalid process models. The above checks can help relate the process models generated by a user to models that an automated approach could generate. Plans or process models that contain errors of unordered or unexecutable steps would never be generated automatically.

Checking Unachieved Preconditions
1. Detect problem with simulation.
(a) Collect failed step(s).
(b) Collect unachieved preconditions of failed step.
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 or incorrect 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 previous steps*:
- Find previous steps that could have produced the needed effects and suggest modifying their role assignment.
(d) *Suggest modifying the step* whose preconditions were not achieved.

Checking Expected Effects
1. Ask user to specify expected effects.
2. Detect problem with simulation.
(a) Collect unachieved effects from each path and record the steps in the failed paths.
3. Help user fix problem.
(a) *Suggest that there are missing steps:*
- Find components in the knowledge base 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 or incorrect 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.

Checking Unordered Steps
1. Detect problem with simulation.
(a) Find unordered substeps by checking interruptions in simulation and unreached steps.
2. Help user fix problem
(a) *Suggest that there are missing ordering constraints*:
- Find the first action in the unreached steps and suggest addition of ordering constraints.

Checking Unexecutable or Effectless Steps
1. Detect problem with simulation.
(a) assertions to be deleted by a step are not true in the situation where the step is executed.
(b) a step produces no effect, i.e., it does not delete or add any assertions.
2. Help user fix problem.
(a) *Suggest modifying role assignments:*
- Find the step's roles that are assigned to wrong objects and suggest modifying them.
(b) *Suggest modifying previous steps:*
- If its previous steps produced assertions that are different from the ones to be deleted, then suggest modifying the previous steps.

Checking Invalid Expressions
1. Detect problem with simulation.
(a) Check the truth/falsity of assertions, including the precondition tests and the expected effect tests.
(b) Find any objects that are tested but undefined.
2. Help user fix problem.
(a) *Suggest modifying invalid expressions.*
(b) *Suggest modifying steps* that access undefined objects (or invalid expressions).

Checking Loops
Inform any loops detected from the simulation.

Checking Disjunctive Branches
Inform any disjunctive branches that are detected from the simulation.

Checking Causal links
Inform any causal links (how some steps generated effects that satisfied the preconditions of some other step) that are detected from the simulation.

Table 3. KANAL verification checks.

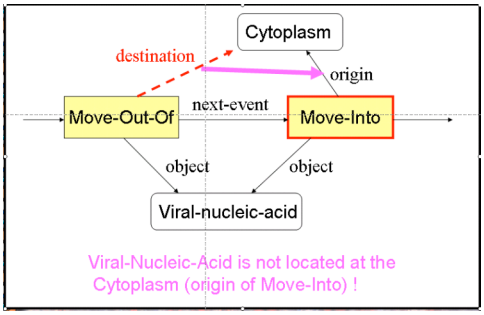

Figure 8: A missing role assignment (missing "destination" link between Move-Out-Of and Cytoplasm) is detected through a failed precondition of the following step.

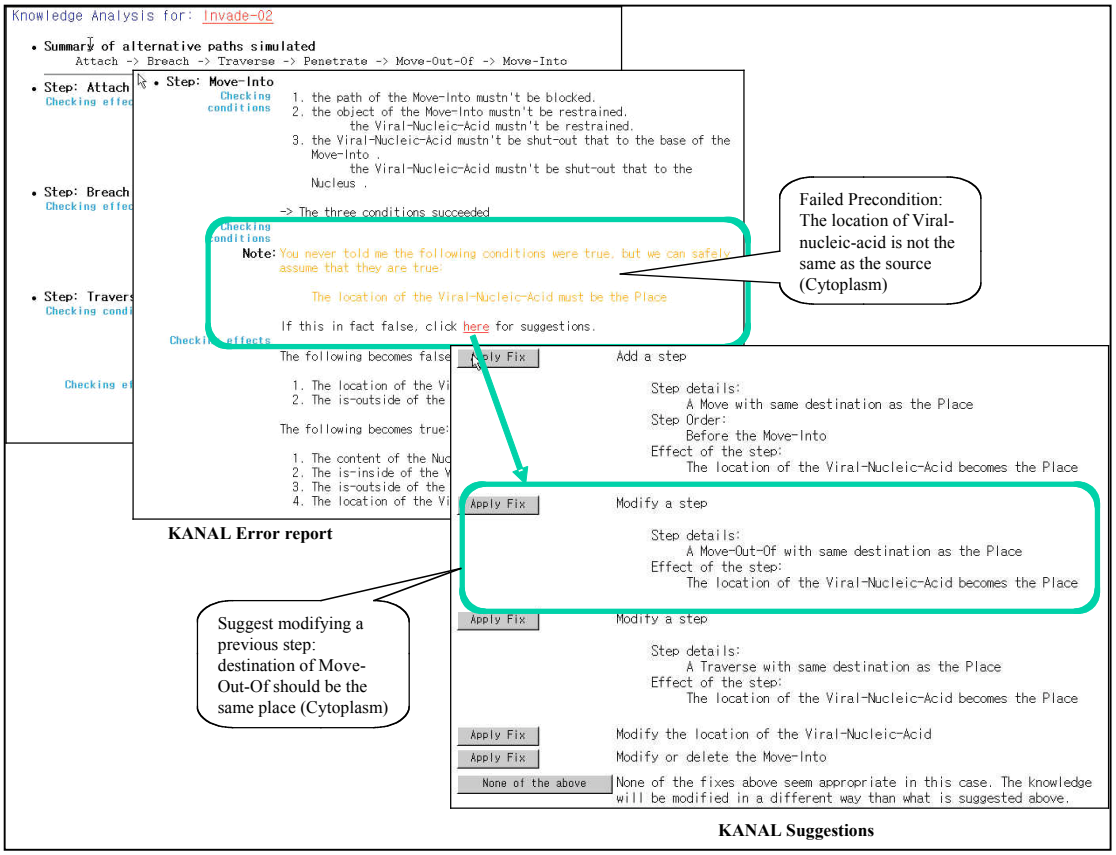

Figure 9: KANAL output

4.5 Example checks made by KANAL

We now show how these checks are made based on knowledge-based descriptions of the components. The highlighted box in Figure 2 shows an example mistake made while building a model of a virus invading a cell. Currently, the destination of the Move-Out-Of step is missing. The editor in SHAKEN prevents users entering invalid links based on the domain and the range of the roles, but such missing information demands a more thorough analysis of the dependencies between the steps. KANAL notices this problem when it checks the precondition of the next step (Move-Into); the Viral-Nucleic-Acid needs to be located at the Cytoplasm in order to Move-Into the Necleus 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 8. 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. (Changing ordering constraints

would be suggested as well if there were other steps that generated the effect.) In fact, the second suggestion directly points to the user's mistake: missing destination link between Move-Out-Of and Eucaryotic-Cytoplasm. Figure 9 shows the KANAL report for this error.

Besides failed preconditions and missing links, KANAL can check unnecessary links, unexecutable steps, effectless steps, causal links among steps, unachieved expected effects, disjunctive branches, and loops, as described above.

4.6 Uses of KANAL by end users

Developing an authoring tool that assists scientists or engineers to directly enter complex process models is a challenging task, and the goal of the overall team project (SHAKEN) that involved many researchers from different research institutes was to develop an end-to-end system that supports such capability. In this study, we were able to measure usages of KANAL in building complex process models and the kinds of checks reported to the user. For the studies of our KA tools in the project, the experimenters needed to spend time and other resources preparing the experiment and analyzing the results for more than a year [Pool et al., 2003]. In general, unlike in some other fields in AI, evaluations of interactive knowledge acquisition tools for capturing procedural knowledge are very hard and rare [Tallis et al., 2001; Cohen et al., 1998]. First, user evaluations are very costly. In areas like machine learning and planning, experiments often amount to running programs repeatedly on already existing test sets. The evaluation of a KA tool requires that a number of subjects spend a fair amount of time doing the study, and for the experimenters to spend time and other resources preparing the experiment (often months) and analyzing the results. The Sisyphus project is an example of the issue discussed above about the intimidating cost of KA evaluations: the limited number of participants can be tracked back to the significant amount of resources required to tackle the knowledge-intensive task that was selected [Shadbolt et al 1999]. Second, most of the research in the field of KA concentrates on knowledge modeling (e.g., how a knowledge engineer models a task domain) and knowledge elicitation (e.g., techniques for interviewing experts). There are few efforts on developing tools for capturing complex procedural knowledge and only some tool developers have conducted usability studies [Tallis et al., 2001; Kim and Gil 2000; Tecuci et al., 2000; Pool et al., 2003]. In many cases, the results are not fully reported in the literature. Third, unless human experiments are carefully designed and conducted, it is hard to draw conclusive results from the data. Often times, the evaluations that test specific claims about a tool or approach are not as thorough or conclusive as we would like to see as scientists, yet these evaluations are very valuable and are shedding some light on topics of interest.

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

Table 4: Uses of KANAL

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

Table 5: Errors and warnings reported
(ratio: number of errors or notes / number of KANAL invocations)

Table 4 shows the number of times KANAL was used during a sequence of two user evaluations. Four biologists participated in the first evaluation and three of them participated again in the second, a smaller scale evaluation. In both evaluations, the biologists created models of processes described in a cell biology textbook [Alberts et al., 1998]. The quality of the resulting knowledge base was assessed with a set of textbook type questions. 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.

Table 5 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, in the second evaluation, 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, the Component Library has been improved over time with richer definitions of actions 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.

KANAL has been also used for military planning and used for critiquing manually built courses of action that are process models of activities to be undertaken by a military unit. A number of tests were run in which two users entered process models (courses of action) in a graphical tool, and tested them with SHAKEN and KANAL [Barker et al., 2003]. The subjects were retired Army officers with no computer science background. This time, they were allowed to extend existing definitions of actions in the background KB when they disagreed with the KANAL reports. 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. For example, a combat power ratio of 3 (blue vs. red) is normally desired for a general military attack, but when an aviation unit attacks an armor unit, a combat power ratio of 0.5 is adequate. Our extension to KANAL facilitates this capability by defining a specialization of an existing action definition with additional preconditions and effects. There were 17 process models tested (each of them has about 20 steps) and the users ran KANAL 74 times. KANAL detected 430 errors or defects in the process models [Kim and Blythe, 2003]. As the users extended existing definitions by creating 17 special cases of actions, they seemed to agree more with KANAL's critiques.

In summary, KANAL checks seemed useful in guiding users in creating process models and checking complex process models, and KANAL produced more useful reports when the knowledge base had richer descriptions of actions.

5. Underlying Principles in CAT and KANAL

As described above, CAT and KANAL focus on different applications with different representations and underlying knowledge bases. Each step in KANAL's process models can have roles instead of input and output data parameters. That is, a link between an introduced object and a step represents how the object plays a particular role for the step. Steps in a process model can be connected via several kinds of links including temporal links and decomposition links. CAT workflows focus on data flow between computational steps and users specify the steps that are involved and link their input and output parameters. Also, CAT components do not have negative effects, and it doesn't need strategies to protect causal links. The CAT algorithm relies on static scanning of the given workflow while KANAL makes use of dynamic simulation in checking process models. However, we found that the users of both

CAT and KANAL share similar requirements and we were able to use similar techniques in assisting them.

First of all, we can map some of CAT's desirable properties to relevant KANAL checks.

- Purposeful: Although KANAL doesn't check whether there are any end results or not, expected effects can be used to determine whether the user specified end results can be actually achieved.
- Grounded: KANAL doesn't have this type of checks since there is no explicit definition of executable components.
- Satisfied: This is similar to KANAL's checks on unachieved conditions.
- Consistent: Although KANAL's checks on causal links are not the same as this, they can be used to determine whether user specified causal links are consistent with simulation results
- Justified: KANAL's checks on unexecutable steps find steps that are not reachable from the first step instead of the end results
- Well-formed: KANAL doesn't have to check this problem since the editor doesn't allow any invalid steps or links.
- Acyclic: This is similar to KANAL's checks on loops in process models
- Parsimonious: KANAL's checks on disjunctive branches report multiple links to the same input. However this is not necessarily a problem in process models.

That is, these set of properties seem useful for checking process models and workflows that are authored by end users, including scientists and military officers.

As we mapped CAT's desirable properties to KANAL checks above, the steps in ErrorScan in Table 2 can be mapped to the KANAL verification functions in Table 3. Although CAT relies on static analysis of workflow templates instead of dynamic simulation and actual checks made are different, they share many similar checks. For example, the KANAL's step for checking unachieved preconditions and expected effects can be mapped to the ErrorScan's step for checking unsatisfied parameters. Table 6 summarizes some of the verification checks made by CAT and KANAL.

In supporting the validation checks, both systems heavily rely on background knowledge including library of actions (or workflow components) and objects that can play certain roles in the process models. In general, domain ontologies and upper or middle level ontologies are commonly used to represent this kind of background knowledge. With this context, the tools become much more helpful in checking that the process model makes sense within the background knowledge that it has.

check types	KANAL: checking process models with dynamic simulation		CAT: checking workflow templates with static scanning	
	Checks	Proposed fixes	Checks	Proposed fixes
Preconditions	Unachieved preconditions during simulation, unexecutable steps,	Add step, modify links of previous or current step, modify ordering constraints	Unsatisfied input parameters	Add steps, add links
Effects	Steps producing no effects	Modify links of previous or current step	Unjustified steps where steps are not connected to end results	Add links
Goals	Unachieved expected results	Add step, modify links, modify ordering constraints	Whether end results exist	Add (end result) step
Step ordering	Missing ordering, loops	Add or modify links	Loops	Remove links in the loops
Step decompositio n	missing first subevent, missing ordering between subevents	Add or modify links	n/a	n/a
Causal links	Whether user specified causal links are consistent with simulation results	Check the causal links that were found	Inconsistent links, whether an output from a step can be used as an input of another	Interpose steps or remove links
Step groundedness	n/a	n/a	Whether the step is executable	Specialize the step
Disjunctive branches	Whether there are disjunctive branches	Check the disjunctive branches that were found	n/a	n/a
Redundancies	Loops	Check the loops that were found	Redundant links, loops	Remove links
Valid steps/links	n/a	n/a	Whether there are steps with undefined components or links to undefined inputs/output parameters	Remove invalid steps or links

Table 6: Similar verification checks made by CAT and KANAL

As indicated by our studies of process models that are authored by end users [Kim and Blythe 2003, Pool et al., 2003; Maechling et al., 2005], checks made by CAT and KANAL seem useful for helping end users. In particular, consistency checks on failed conditions and inconsistent links, and completeness checks on unnecessary ordering and unsatisfied parameters seem to cover many user errors.

However, users seem to need additional proactive capabilities for user assistance. The following list summarizes some of these additional requirements.

- System did not tell the users clearly enough what the system can do now and what it needs to know.

- Users need assistance in determining whether the system was drawing the kinds of inference intended. Users want to perform such checks regularly.
- Users need help to keep on track: manage work in progress and remove unwanted work.
- Users have difficulty with:
    - Starting new process models or tasks.
    - Entering complex procedural knowledge with nested steps and/or nested roles among objects involved.
    - Comparing and assessing similarity between authored knowledge and prior knowledge.
    - Facilitating consistency checking across user efforts.
    - Establishing confidence and competence on knowledge entered.

We are currently developing a front-end KA dialogue tool that supports some of these capabilities using a set of acquisition goals and strategies for them [Kim and Gil 2007; Gil and Kim 2002].

6. Related Work

Knowledge acquisition and knowledge engineering remain challenging areas for many scientific computations and business-related AI applications. Many knowledge acquisition approaches have been targeting knowledge engineers [Wielinga et. al., 1992; Yost 1992; Schreiber et al., 1999; Fikes et al.,1997], and those approaches that have been developed for end users (i.e., users who do not have computer science backgrounds) [Eriksson et al. 1995; Marcus and McDermott, 1989] only allow them to specify certain kinds of knowledge, i.e., domain-specific knowledge regarding instances and classes. Some systems use a variety of elicitation techniques to acquire descriptive knowledge [Clark, et al., 2001, Gaines and Shaw, 1993, Shadbolt and Burton, 1989], often in semi-formal formats. Recently there has been increasing interest in tools that enable end users to enter complex procedural knowledge. Some systems use rich background knowledge and graphical interfaces [Clark et al., 2001]. Other tools focus on detecting gaps and errors in the knowledge specified by the user [Blythe et al., 2001]. Some tools combine interactive dialogue approaches with domain ontologies and upper or middle level ontologies [Witbrock et al., 2003]. Alternative approaches apply learning and induction techniques to examples provided by users in a natural way as they are performing tasks [Mitchellet al.,1985, Cypher 1993, Bareiss et al., 1989, Lau et al., 2003]. Although these tools may be more accessible to end users, they are only useful in circumstances where users can provide a variety of examples. When examples are not readily available, we may need KA tools for direct authoring. Our systems present a knowledge-based approach that uses planning techniques in order to guide users in generating complete and consistent process models or workflow templates.

Past research in validation and verification of knowledge bases addresses the detection of errors in rule bases [Preece and Shinghal, 1994; O'Keefe and O'Leary, 1994] or in ontologies [McGuinness et al., 2000; Noy et al., 2006] and has not addressed process models specifically.

Some work on web service composition provides approaches to match and select related semantic web services in order to compose workflows [Sirin et al., 2003; Burstein et al., 2000, Sycara et al, 1999]. However, existing tools provide limited support in generating correct and complete end-to-end compositions, and do not address the user interaction issues raised in our applications. Existing approaches for composition of web services [Chen et al., 2003, McDermott 2002; Narayanan and McIlraith 2002; McIlraith and Son 2001; Sheshagiri, et al., 2003; Thakkar et al. 2000] use expressive languages and sophisticated reasoning and planning techniques to generate valid compositions of services. They complement our work in that they do not address user interaction issues. The Web Services ToolKit (WSTK) [Srivastava 2000] includes a composition engine, but it has very limited models of the data used by the services, which limits the support that underlying reasoners can provide. SWORD [Ponnekanti and Fox 2002] is a toolkit that addresses interactive service composition. However, it is designed for developers who have programming skills, not for the end users that our work is intended for.

There have been some interactive approaches proposed for planning applications. PASSAT and similar systems [Myers 1997, Myers et al., 2002, Myers et al., 2003] take plan sketches provided by a user and interpret and complete the sketches using domain knowledge in terms of hierarchical task network (HTN) schemas. PASSAT detects errors caused by extra steps in the sketches and violated conditions. Our work is complementary in that it utilizes a component-based approach rather than HTN. The Advisable Planner and Advisable Agents frameworks [Myers 1996, Myers 2000] use grammars to express advice in terms that automated planners (or agents) can use, and detects conflicting advice provided by the user. Our approach is complementary in that these kinds of preferences could be used to narrow down the suggestions provided by our algorithms.

Interactive algorithms to guide the search of solutions have used visualization techniques to explore the tradeoffs among solutions [Anderson et al 2000, Blythe 2002]. We do not address how to guide the user through the solution space, but in future work we would like our system to take a more proactive role in helping the user understand how their choices of components and links affect the solutions that will result.

Other related work addresses the acquisition of planning knowledge. Approaches to develop tools to aid users to specify planning domains [Aler and Borrajo 2002, Kim and Blythe 2003, McCluskey et al., 2003] are complementary to our work in that we assume that the specification of components is provided to our system, which exploits that knowledge to form the workflows. Other work looks at different

planning tasks from a knowledge engineering perspective [Benjamins et al., 96], where the planner itself is configured from problem solving methods, while our work investigates the composition of the workflows (or plans) themselves. Research on interactive tools to acquire planning knowledge is also related [Chien, 1998; Myers, 1996; Huffman and Laird, 1995], but their focus is on acquiring knowledge about how to generate plans instead of acquiring the specific plans themselves. TRAINS [Ferguson et al 95] uses dialogue-based techniques to determine the correctness and relevance of planning knowledge added by the user, including planning tasks and goals. Our approach is complementary in that it incorporates user input into the workflow through a limited set of actions, and uses properties of desirable process models to follow up with the user in terms of possibly incorrect portions of it.

Description logics have been used in various aspects of planning, but not to guide interactive generation of plans. CLASP used description logics to reason about planning states and actions in order to relate procedures by exploiting action taxonomies [Devanbu and Litman 92]. Description logics have also been used to guide plan generation and plan recognition by reasoning about subsumption of plan structures [Wellman, 1990, Alterman, 1986, Litman, 1994].

7. Conclusions and Future Work

Our work has been motivated by the requirements of different KA applications including authoring process models in biology, manual development of military plans and composing computational workflows in earthquake science. Based on their requirements, we have developed two authoring tools, CAT and KANAL, with which users can author and check process models and workflows, and call on the system to provide intelligent assistance. KANAL was driven by the needs to verify user entered process models in biology and to critique manually built military plans. Users of KANAL, including retired Army generals, wanted to assess potential flaws in the plans and process models, and easily understand how they could improve existing process models. Engineers and earthquake scientists needed a tool with which they could compose a workflow flexibly following different authoring strategies, since their constraints on workflows are not pre-defined and tend to become more explicit while they explore different selections of components and their connections. While designing these tools and analyzing their problems, we have identified common requirements that the systems have and developed an approach to fulfill these requirements. Our work combines knowledge-based representation of components, together with planning techniques that can track the relations and constraints among components, no matter the order of the user's actions in specifying the plan or workflow. For KANAL we have defined a set of checks that the system performs to verify user entered plans. CAT has been developed more recently and defines a set of formal properties for correct workflow templates. The ErrorScan algorithm guides users to specify correct workflows using these properties.

Both CAT and KANAL are driven by AI planning techniques, and their checks are similar. Although the systems support different interfaces and the interactions are driven by specific needs and system constraints (e.g. use of KM simulation in KANAL vs. stepwise checks in CAT), the same techniques seem useful for both of them. We believe that our techniques could be useful for other similar process model authoring problems.

Verification of user entered workflows or process models is useful in other contexts. For example, users may compose workflows using editors off-line, then invoke a workflow verification to report problems with the workflow. This is typical in scientific environments today, where scientists use text editors to create workflows. There are workflow editors that provide useful graphical capabilities but have no comprehensive error checking facilities. Our workflow verification techniques would be a useful addition to these workflow editors. Another context in which workflow/plan verification would be beneficial is reuse, adaptation, and merging of previously existing workflows or plans. In scientific applications, retrieval of past successful workflows as a starting point to design new ones is commonplace. Our workflow verification techniques can help a scientist during the process of adapting these workflows to the new situations. Likewise, existing process models can be used to specify similar but different process models (e.g. a variation of Lambda virus invasion). Finally, workflow verification techniques would be useful in assisting users to develop end-to-end applications by merging previously existing workflows that address smaller aspects of the overall application. In merging workflows or process models, many inconsistencies, gaps, and overlaps may occur. Ultimately, user-guided composition would involve not only interactive development but also the aforementioned modalities of one-shot editing, retrieval and adaptation, and merging of existing workflows or process models.

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 system's background knowledge used to check their plans is inconsistent with the user's knowledge, or 3) the analysis results are not clearly presented in the interface. As described earlier, KANAL supports a simple way in which the user can add more detail to an action to cover slightly different behavior in special cases. We are looking at other potential ways in which background knowledge can be semi-automatically extended based on user feedback.

An important extension of our approach would be to guide the user through the workflow generation space in a systematic manner, so that the user explores each area of the search space only once. This would involve remembering the workflows that the user has composed, and constraining the possible future actions that the user can take to extend the workflow by taking into account areas of the search space already explored.

The interactive approach could provide a complementary capability to automatic techniques. For example, the verification results may enable integration of interactive and automatic techniques for plan development. After a user sketches a plan, an automated planner could fill in the details and missing steps. However, in order for this to work, it is necessary to ensure that errors in the plan created by a user, such as redundant steps or inconsistent links, are removed before an automated planner takes it and starts expanding it with more detail.

8. Acknowledgements

We would like to thank earthquake scientists, biologists, and military officers who participated in our studies. We would also like to thank our colleagues in the SCEC-CME and SHAKEN projects for valuable discussions and integration support. This research was funded by the National Science Foundation (NSF) with award numbers EAR-0122464 and CCF-0725332, and by the DARPA Rapid Knowledge Formation (RKF) program under contract number N66001-00-C-8018.

References

Alberts, B., Bray, D., Johnson, A.,Lewis, J., Raff, M., Roberts, K. & Walter, P., 1998. *Essential Cell Biology: An Introduction to the Molecular Biology of the Cell*. Garland Publishing Inc.

Aler, R. & Borrajo, D., 2002. On Control Knowledge Acquisition by Exploiting Human-Computer Interaction, *Proceedings of International Conference on AI Planning and Scheduling.*

Alterman, R., 1986. An Adaptive Planner, *Proceedings of National Conference on Artificial Intelligence.*

Altintas, I., C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, & S. Mock, 2004. Kepler: Towards a Grid-Enabled System for Scientific Workflows, *Workflow in Grid Systems Workshop in GGF10 - The Tenth Global Grid Forum*.

Anderson, D., Anderson, E., Lesh, N.B., Marks, J.W, Mirtich, B., Ratajczack, D.; Ryall, K., 2000. Human-Guided Simple Search. *Proceedings of National Conference on Artificial Intelligence.*

Barker, K., Blythe, J., Borchardt, G., Chaudhri, V., Clark, P., Cohen, P., Fitzgerald, J., Forbus, K., Gil, Y., Katz, B., Kim, J., King, G., Mishra, S., Morrison, C., Murray, K., Otstott, C., Porter, B., Schrag, R., Uribe, T., Usher, J. & Yeh, P., 2003.Knowledge Acquisition Tool for Course of Action Analysis. *Proceedings of the Innovative Applications of Artificial Intelligence.*

Bareiss, R., Porter, B., & Murray, K. 1989. Supporting start-to-finish development of knowledge bases, *Machine Learning* 4:259–283.

Barker, K., Porter, B., & Clark, P**.** 2001. A library of generic concepts for composing knowledge bases, *Proceedings of International Conference on Knowledge Capture.*

Benjamins, R., Barros, L., & Valente, A., Constructing Planners Through Problem-Solving Methods, *Proceedings of the Knowledge Acquisition Workshop*, 1996.

Blythe, J., 2002. Acquisition and Visualization of Planning Preferences and Constraints. *Proceedings of National Conference on Artificial Intelligence.*

Blythe, J., Deelman, E., Gil, Y., Kesselman, C., Agarwal, A., Mehta, G., & Vahi, K., 2003. The Role of Planning in Grid Computing, *Proceedings of the International Conference on Automated Planning and Scheduling.*

Blythe, J. Kim, J., Ramachandran, S., & Gil, Y., 2001. An Integrated Environment for Knowledge Acquisition. *Proceedings of the International Conference on Intelligent User Interfaces*.

Burstein, M, McDermott, D., Smith, D., & Westfold, S, 2000. Derivation of Glue Code for Agent Interoperation. *Proceedings of the Fourth International Conference on Autonomous Agents*.

Chen, L., Shadbolt, N., Goble, C.A., Tao, F., Cox, S. J., Puleston, C., & Smart, P. R., 2003. Towards a Knowledge-Based Approach to Semantic Service Composition, *International Semantic Web Conference*, pp. 319-334.

Chien, S., & Mortensen, H., 1996. Automating Image Processing for Scientific Data Analysis of a Large Image Database, IEEE *Transactions on Pattern Analysis and Machine Intelligence* 18 (8): pp. 854-859.

Chin Jr, G., Leung, L. R., Schuchardt, K., & Gracio, D., 2002. New paradigms in problem solving environments for scientific computing. *Proceedings of Intelligent User Interfaces 2002*, pp. 39-46.

Churches, G., Harrison, A., Maassen, J., Robinson, C., Shields, M., Taylor I., & Wang, I., 2005. Programming Scientific and Distributed Workflow with Triana Services, *Grid Workflow 2004 Special Issue of Concurrency and Computation: Practice and Experience.*

Clark, P. & Porter, B., 2005. The knowledge machine. In http://www.cs.utexas.edu/users/mfkb/km.html.

Clark, P., Thompson, J., Barker, K. and Porter, B., Chaudhri, V., Rodriguez, A., Thomere, J., Mishra, S., Gil, Y., Hayes, P. & Reichherzer, T., 2001. Knowledge Entry as Graphical Assembly of Components, *Proceedings of International Conference on Knowledge Capture*.

Cohen, P., Schrag, R., Jones, E., Pease, A., Lin, A., Starr, B., Gunning D., & Burke, M. 1998. The DARPA High-Performance Knowledge Bases Project. *AI Magazine,* 19 (4).

Cypher, A. 1993. *Watch what I do: Programming by demonstration*. MIT Press.

Deelman, E. Callaghan, S. Field, E. Francoeur, H. Graves, R. Gupta, N. Gupta, V. Jordan, T.H. Kesselman, C. Maechling, P. Mehringer, J. Mehta, G. Okaya, D. Vahi, K. & Zhao, L. , 2006. Managing Large-Scale Workflow Execution from Resource Provisioning to Provenance tracking: The CyberShake Example, *Proceedings of the e-Science*.

Deelman, E., Singh, G., Su, M., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G.B., Good, J., Laity, A.,. Jacob, J.C., & Katz, D.S., 2005, Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems, *Scientific Programming Journal*, Vol 13(3), Pages 219-237.

Deelman, E. Ewa Deelman, Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Patil, S., Su, M., Vahi, K., & Livny, M., 2004. Pegasus: Mapping Scientific Workflows onto the Grid, *Across Grids Conference*, 2004.

Devanbu, P, & Litman, D., 1996. CLASP - a plan representation and classification scheme for a software information system, *Artificial Intelligence,* 84, 1-35.

Eriksson, H., Shahar, Y., Tu, S.W., Puerta, A.R. & Musen, M., 1995. Task modeling with reusable problem-solving methods, *Artificial Intelligence*, 79, 293-326.

Ferguson, G., Allen, J., & Miller, B., 1995. TRAINS-95: Towards a Mixed-Initiative Planning Assistant *Proceedings of International Conference on Planning and Scheduling*.

Fikes, R.; Farquhar, A., & Rice, J. 1997. Tools for assembling modular ontologies in Ontolingua, *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 436–441.

Fikes, R. & Nilsson N., 1971. STRIPS: a new approach to the application of theorem proving to problem solving, *Artificial Intelligence*, 2:189-208.

Gaines, B.R. & Shaw, M., 1993. Knowledge acquisition tools based on personal construct psychology, *The Knowledge Engineering Review*, *8* (1). 49-85.

Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D. & Wilkins, D., 1998. *PDDL - the planning domain definition language*, Technical report, Yale University.

Gil, Y, Ratnakar, V., Deelman, E., Spraragen, M., & Kim, J., 2007. Wings for Pegasus: A Semantic Approach to Creating Very Large Scientific Workflows, *Proceedings of the Nineteenth Innovative Applications of Artificial Intelligence Conference*.

Gil, Y. & Kim, J., 2002. Interactive Knowledge Acquisition Tools: A Tutoring Perspective, *Proceedings of the 24th Annual Meeting of the Cognitive Science Society (COGSCI-2002)*, pp. 357-362.

Kim, J. & Blythe, J., 2003. Supporting Plan Authoring and Analysis. *Proceedings of International Conference on Intelligent User Interfaces.*

Kim, J., Deelman, E., Gil, Y., Mehta, G., & Ratnakar, V., 2007, Provenance Trails in the Wings/Pegasus System, *Journal of Concurrency And Computation: Practice And Experience*, November.

Kim J. & Gil, Y., 2007, Incorporating Tutoring Principles into Interactive Knowledge Acquisiiton, *International Journal of Human-Computer Studies*, Volume 65, Issue 10, Pages 852-872.

Kim J. & Gil, Y., 2004. Towards Interactive Composition of Semantic Web Services, *Proceedings of the AAAI Spring Symposium on Semantic Web Services*.

Kim J. & Gil, Y., 2003. Proactive Acquisition from Tutoring and Learning Principles, *Proceedings of the AI in Education* , 2003.

Kim, J. & Gil, Y 2001. Knowledge Analysis on Process Models, *Proceedings of International Joint Conference on AI.*

Kim, J., Spraragen, M., & Gil, Y. 2004, An Intelligent Interface for Web Service Composition, *Proceedings of Intelligent User Interfaces Conference.*

Kim, J., Spraragen, M., & Gil, Y. 2005, Interactive Workflow Composition as Plan Synthesis, *Internal project report,* 2005.

Lansky, A., Friedman, M., Getoor, L., Schmidler, S., & Short, N., 1995. The COLLAGE/KHOROS Link: Planning for Image Processing Tasks, *AAAI Spring Symposium on Integrated Planning Applications*.

Lau, T., Wolfman, S., Domingos, P., & Weld, D., 2003. Programming by Demonstration using Version Space Algebra, *Machine Learning,* 53(1).

Litman, D., 1994. Plan Recognition through Description Logics, *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning.*

MacGregor, R., 1992. *LOOM Users Manual*, Technical Report No. ISI/WP-22, USC/Information Sciences Inst.

Malone, T. W., Crowston, K. & Herman, G. A..,(2003). *Organizing Business Knowledge: The MIT Process Handbook*, MIT Press.

Marcus, S. & McDermott, J., 1989. SALT: A knowledge acquisition language for propose-and-revise systems. *Artificial Intelligence*, 39(1):1--37.

Maechling, P., Chalupsky, H., Dougherty, M., Deelman, E., Gil, Y., Gullapalli, S., Gupta, V., Kesselman, C., Kim, J., Mehta, G., Mendenhall, B., Russ, T., Singh, G., Spraragen, M., Staples, G., & Vahi,. K., 2005. Simplifying Construction of Complex Workflows for Non-Expert Users of the Southern California Earthquake Center Community Modeling Environment, *ACM SIGMOD Record, special issue on Scientific Workflows*, 34(3).

McCluskey, L., Liu D., & Simpson R., 2003. GIPO II: HTN Planning in a Tool-supported Knowledge Engineering Environment, *Proceedings of the International Conference on Automated Planning and Scheduling*.

McDermott, D., 2002 Estimated-Regression Planning for Interactions with Web Services, *Proceedings of AI Planning Systems Conference*.

McGuinness, D., Fikes, R., Rice, J. & Wilder, S., 2000. An Environment for Merging and Testing Large Ontologies, *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*.

McIlraith, S. & Son, T., 2001. Adapting GOLOG for programming in the semantic web, *Fifth International Symposium on Logical Formalizations of Commonsense Reasoning*.

Mitchell, T., Mahadevan, S., & Steinberg, L. 1985. LEAP: A learning apprentice for VLSI design. *Proceedings of the 1985 International Joint Conference on Artificial Intelligence.*

Myers, K., 1996. Advisable Planning Systems, *Advanced Planning Technology.* AAAI Press.

Myers, K., 1997. Abductive Completion of Plan Sketches, *Proceedings of National Conference on Artificial Intelligence*.

Myers, K., 2000. Domain Metatheories: Enabling User-Centric Planning. *Proceedings of the AAAI Workshop on Representational Issues for Real-World Planning Systems*.

Myers, K., Jarvis, P., Tyson, M., & Wolverton, M., 2003. A Mixed-Initiative Framework for Robust Plan Sketching*, Proceedings of International Conference on Automatic Planning and Scheduling.*

Myers, K. L., Tyson, W. M., Wolverton, M. J., Jarvis, P. A., Lee, T. J., & desJardins, M., 2002. PASSAT: a user-centric Planning Framework, *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*.

Narayanan, S., & McIlraith, S. 2002. Simulation, verification and automated composition of web services, *Proceedings of the 11th International World Wide Web*.

Nau, D. 2007. Current trends in automated planning, *AI Magazine,* 28(4):43–58.

Noy, N. Chugh, A. Liu, W., & Musen M. A. 2006. A Framework for Ontology Evolution in Collaborative Environments, *Proceedings of the Fifth International Conference on the Semantic Web.*

O'Keefe, R. & O'Leary, D., 1993. Expert system verification and validation, *Expert Systems with Applications: An International Journal*, 6(1): 57-66.

OWL, 2008. http://www.w3.org/TR/owl-features/, 2008

Paolucci, M., Kawmura, T., Payne, T., & Sycara, K. 2002. Semantic matching of web services capabilities. *First International. Semantic Web Conference*.

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

Pool, M., Murray, K., Fitzgerald, J., Mehrotra, M., Schrag, R., Blythe, J., Kim, J., Chalupsky, H., Miraglia, P., Russ, T. & D.Schneider, 2003. Evaluating SME-Authored COA Critiquing Knowledge, *Proceedings of the International Conference on Knowledge Capture*.

Sacerdoti, E. 1977. *A Structure for Plans and Behavior*, New York: Elsevier.

Shadbolt, N. & Burton, A.M., 1989. The empirical study of knowledge elicitation techniques. *SIGART Newsletter*, 108. 15-18

Shadbolt, N., O'hara, K. & Crow, L., 1999. The experimental evaluation of knowledge acquisition techniques and methods: history, problems and new directions, *International Journal of Human-Computer Studies,* 51.

Sheshagiri, M., desJardins, M., & Finin, T. 2003. A planner for composing services described in daml-s, *ICAPS 2003 Workshop on Planning for Web Services Program*.

Schreiber, G., Akkermans, H., Anjewierden, A., Hoog, R.d., Shadbolt, N., Velde, W.V.d. & Wielinga, B., 1999. *Knowledge Engineering and Management The CommonKADS Methodology*, MIT Press.

Sirin, E., Parsia, B., & Hendler, J., 2004.: Filtering and Selecting Semantic Web Services with Interactive Composition Techniques, *IEEE Intelligent Systems,* 19(4): 42-49.

Srivastava, B. 2000. Web services toolkit (WSTK).

Sycara, K. Klusch, M., Widoff, S., & Lu. J., 1999. Dynamic Service Matchmaking among Agents in Open Information Environments. *ACM SIGMOD Record, Special Issue on Semantic Interoperability in Global Information Systems*.

Tallis, M., Kim J., & Gil, Y., 2001. User Studies Knowledge Acquisition Tools: Methodology and Lessons Learned, *Journal of Experimental and Theoretical Artificial Intelligence, Vol 13. No 4.*

Tate, A. 1977. Generating project networks, *Proceedings of the International Joint Conference on Artificial Intelligence.*

Tate, A. 1996. Representing plans as a set of constraints – the <i-n-ova> model, *Proceedings of the International Conference on AI Planning and Scheduling.*

Taylor, I., Deelman, E. Gannon, D.B., & Shields, M. (Eds.) 2006. *Workflows for e-Science: Scientific Workflows for Grids*, Springer Verlag.

Tecuci, G., Boicu, M., Marcu, D., Bowman, M., Ciucu, F., & Levcovici, C., 2000. Rapid Development of a High Performance Knowledge Base for Course of Action Critiquing, *Proceedings of the National Conference on Artificial Intelligenc.*

Thakkar, S., Knoblock, C., Ambite, J., & Shahabi, C. 2000, Dynamically composing web services from on-line sources, *Proceedings of the AAAI Workshop on Intelligent Service Integration*.

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

Voit, E.O. 2000. *Computational Analysis of Biochemical Systems*. Cambridge: Cambridge University Press.

Weld, D. 1999, Recent Advances in AI Planning, *AI Magazine,* 20(2).

Wellman, M., 1990. *Formulation of Tradeoffs in Planning Under Uncertainty*, PhD thesis, MIT.

Wielinga, B. J.; Schreiber, A. T.; & Breuker, A. 1992. KADS: a modelling approach to knowledge acquisition, *Knowledge Acquisition* 4(1):5–54.

Witbrock, M. J., Baxter, D., Curtis, J., Schneider, D., Kahlert, R., Miraglia, P., Wagner, P., Panton, K., Matthews, G., & Vizedom, A., 2003.An Interactive Dialogue System for Knowledge Acquisition in Cyc. 138-145. *Proceedings of the IJCAI-03 Workshop on Mixed Initiative Intelligent Systems*.

Wroe C., Goble, C.A., Greenwood, M., Lord, P., Miles, S., Papay, J., Payne, T., & Moreau, L.2004. Automating Experiments Using Semantic Data on a Bioinformatics Grid, *IEEE Intelligent Systems special issue on e-Scien*ce Jan/Feb 2004.

Yang, Q. 1990. Formalizing planning knowledge for hierarchical planning. *Computational Intelligence* 6(1):12–24.

Yost, G. R. 1992. *TAQL: A Problem Space Tool for Expert System Development*. PhD thesis, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA.