A Knowledge-Based Approach to Interactive Workflow Composition

Jihie Kim, Yolanda Gil, Marc Spraragen

University of Southern California/Information Sciences Institute
Marina del Rey, CA 90292 USA
+1 310 448 8769
{jihie, gil, marcs}@isi.edu

ABSTRACT

Complex applications in many areas, including scientific computations and business-related web services, are created from collections of components to form computational workflows. In many cases end users have requirements and preferences that depend on how the workflow unfolds, and that cannot be specified beforehand. Workflow editors therefore need to be augmented with intelligent assistance in order to help users in several key aspects of the task, namely: 1) keeping track of detailed constraints across selected components and their connections; 2) accommodating flexibly different strategies to construct workflows; e.g., from general knowledge of necessary tasks, from desired results, or from available data; and 3) taking partial or incomplete descriptions of workflows and understanding the steps needed for their completion. We have developed a system called CAT (Composition Analysis Tool) that analyzes workflows and generates error messages and suggestions in order to help users compose complete and consistent workflows. Our approach combines knowledge bases, which have rich representations of components and constraints, together with planning techniques that can track the relations and constraints among individual components. We have formalized our approach based on AI planning principles, allowing us to formulate claims about the underlying algorithms as well as the resulting workflows.

Keywords

workflow composition, description logic, interactive approach

INTRODUCTION

Composing computational workflows is essential in many areas, including scientific computing and business applications. For example, scientists have growing needs to dynamically produce computation workflows where they assemble and link various models that address different aspects of the phenomenon under study [Griphyn 03, SCEC 03, Geodise 03, MyGrid 03]. In business applications, web services are becoming a promising framework for composing new applications out of existing software components (such as software modules or web services). Some planning approaches have been used in this context. [Lansky et al 95, Chien et al 96] However, automatic planning approaches are not always appropriate to generate these workflows. In some cases, users may not have explicit descriptions of the desired end results or goals in the beginning. Users may only have high-level or partial/incomplete descriptions of the desired outcome or the initial state, and the real goals and initial data input may become clear as they see the features of the components that can be used. Business agreements and past experiences of how the components work may also affect the development of the workflow.

The goal of our work is to develop interactive tools for composing workflows where users select and configure components and the system assists the users by detecting errors and providing intelligent suggestions to solve them. This provides a complementary capability needed for developing computational workflows.

This paper presents an approach to interactive workflow construction. Novel features of the approach include: 1) combining description logic and planning frameworks; 2) interactive workflow construction through planning techniques; 3) properties for verification of manually created workflows based on planning techniques; and 4) an algorithm that assists users by enforcing those properties and suggesting possible next steps.

Using this approach, we have developed the CAT system to analyze a partial workflow composed by the user, notify the user of issues to be resolved in the current workflow, and suggest to the user what actions could be taken next. In this paper we focus on the planning techniques that are used in our framework. The details on our system's interfaces and user interactions are described in [Kim et al 04].

The paper begins by describing our motivations and goals based on a scientific application (earthquake simulation) which led us to develop CAT. We outline the representation that we have developed to describe workflow components. We define desirable properties of workflows, and the composition actions that the user can employ to refine the workflow. We then present the

ErrorScan algorithm that analyzes a partial workflow, and generates suggestions that will lead the user to compose an error-free workflow. Finally, we present CAT's contributions in context of related work.

Motivation

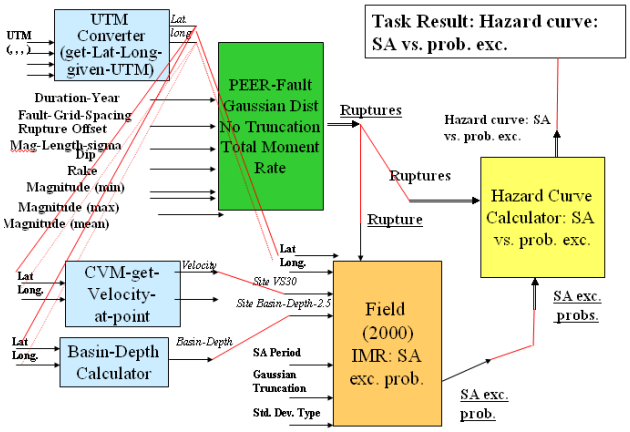

Figure 1. An example computational workflow in the earthquake science domain.

Scientific progress is significantly accelerated by integrating components that model different aspects of the phenomena being studied. Projects investigating large-scale scientific applications include [SCEC 03, GriPhyN 03,Geodise 03, MyGrid 03]. We take an example from our collaboration with SCEC (Southern California Earthquake Center); similar issues arise in other scientific applications.

Figure 1 shows an example of a completed workflow for from seismic hazard analysis (SHA) to enable 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. The models are complex, heterogeneous, and come with many constraints on their parameters and their use with other models.

In their work, scientists and engineers often want to sketch the workflows themselves, influencing the choices of the software components and the links between them. In many cases end users have requirements and preferences that often depend on how the workflow unfolds and 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.

An important aspect of our approach is verifying workflows by detecting errors and missing steps. In this paper, we focus on the use of these verification techniques in an interactive setting. However, workflow verification 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 also many workflow editors that provide useful graphical capabilities [SmartDraw 03, Khoros 03, BizTalk 03] but have no comprehensive error checking facilities. Other composition tools that use domain knowledge provide limited help in guiding the users during the interaction because they don't explicitly use the semantics associated with steps and links [Chen et al 03, Sirin et al 03]. Our workflow verification techniques would be a useful addition to these workflow editors. Another context in which workflow verification would be beneficial is reuse, adaptation, and merging of previously existing workflows. 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. 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, many inconsistencies, gaps, and overlaps may occur. Ultimately, user-guided workflow composition would involve not only interactive development but also the abovementioned modalities of one-shot editing, retrieval and adaptation, and merging of existing workflows.

Another use of workflow verification is to enable integration of interactive and automatic techniques for workflow development. After a user sketches a workflow, 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 workflow created by a user, such as redundant steps or inconsistent links, are removed before an automated planner attempts to complete the workflow. We look at this aspect in detail later in the paper.

APPROACH

Figure 2 below shows the CAT interface that we built to support interactive workflow composition. The CAT system has been used to support developing computational workflows in earthquake science domain like the one for seismic hazard analysis that is shown in Figure 1.

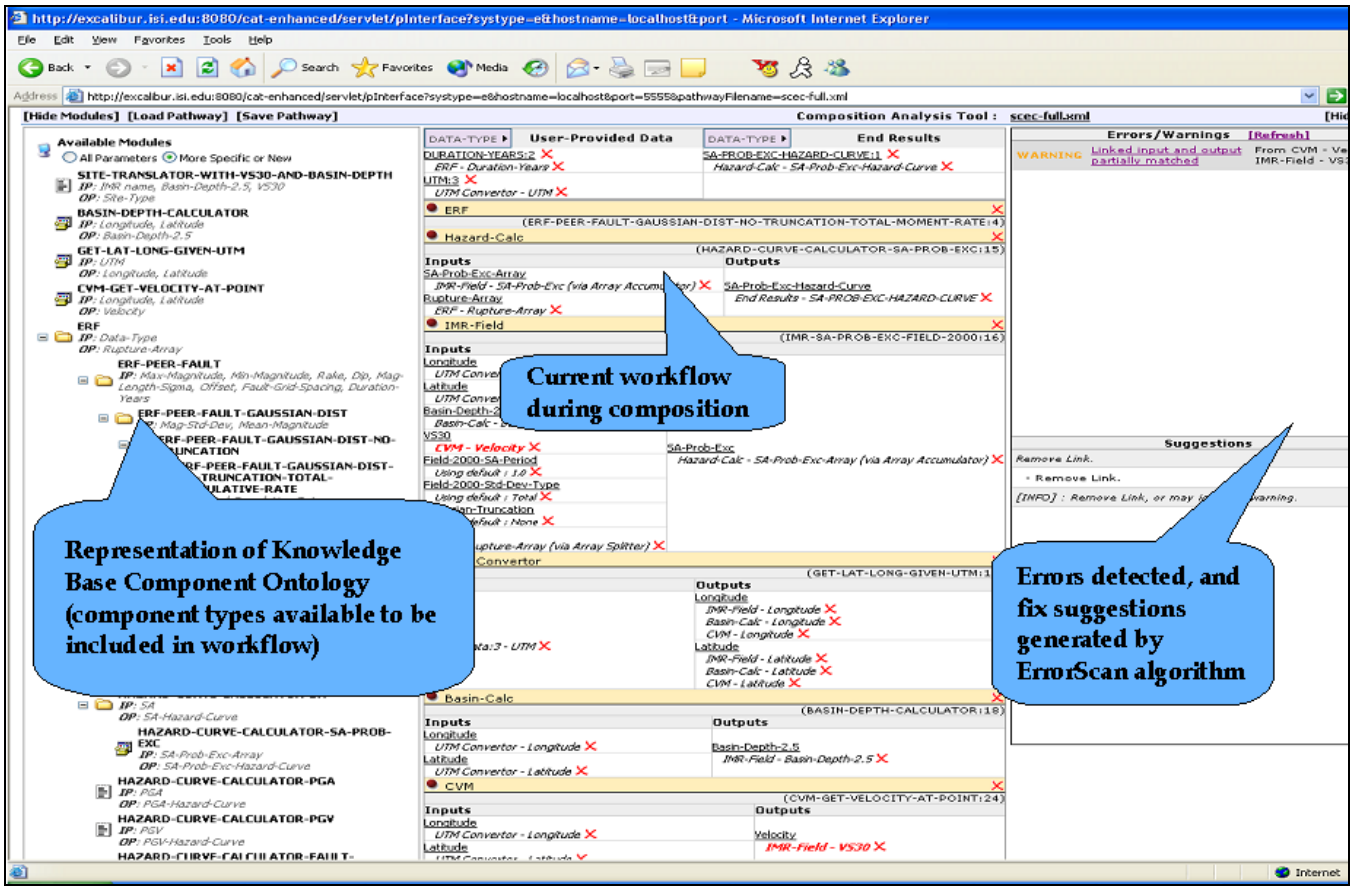

Figure 2. The CAT interface for composing workflows for seismic hazard analysis.

In our approach as implemented in CAT, users form a workflow incrementally, and the system checks the validity of the workflow and suggests what to do next. In the process of constructing a workflow, users can add a component to the workflow and make links between the components. When adding a component, the user may indicate an abstract type of component that will be further specialized at a later step. For example, the user may specify that the workflow will include a fault rupture model, and not decide which model to use until the wave propagation model is selected. The user may start from the end results, or from a set of known data, or from the components they want to include.

The analysis of partial workflows created by the user is done using an AI planning framework [Weld 99]. Each component is treated as a step in the plan, the inputs of a component are the preconditions of that step, and the component's outputs are the step's effects. The links between components 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. Each action taken by the user (add/remove component, specialize component, add/remove link) is akin to a refinement operator in plan generation. While automatic planning systems can explore the space of plans systematically and guarantee that the final plans are correct, interactive workflow composition requires an approach that lets the user decide what parts of the search space to explore and that can handle incorrect partial workflows.

The next subsection describes how we represent components, abstract types of components, and their constraints.

Supporting Knowledge Base

Figure 3 shows a portion of a sample CAT Knowledge Base (KB) for a travel domain. In the KB's Domain Ontology, there is a hierarchy of data types represented in Loom classes [MacGregor 90]. In the Component Ontology, each component is represented in terms of its input and output parameter data types. For example, component Car-Rental-by-Airport needs an airport (as arrival-place) and a Date (as arrival-date) as input, and produces a Car-Reservation. 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. Since each parameter of the component is defined in terms of the data types in the Loom KB, CAT can exploit Loom's reasoning capabilities and use them in helping users construct correctly formulated workflows.

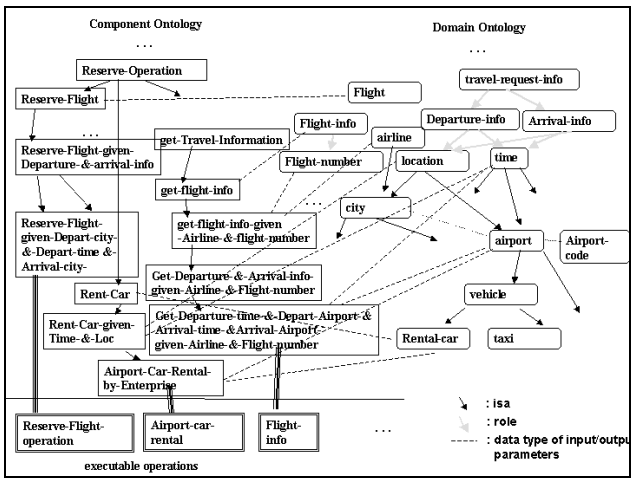

Figure 3. An example CAT Knowledge Base.

The knowledge base supports the following queries:

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

Examples:

- *input-parameters* (Rent-Car) = {arrival-time arrival-location}.
- *range* (Rent-Car, arrival-time) = time.
- *subsumes* (Rent-car, Airport-Car-Rental)= true.
- *subsumes* (airport, location) = false.

WORKFLOW

A workflow W is a tuple <*C, L, I, G*> where *C* is a set of workflow components, *L* is a set of *links, I* is a set of *Initial-Input* components and *G* (for Goals) is a set of *End-Result* components. Initial-Input components and End-Results are handled in most respects as any other component. Initial-Input components (user-input data) can be handled as components with one output parameter and no input parameters. End-Result components are components with one input and no outputs.

Each *link* is a tuple $<c_o,p_o,c_i,p_i>$ where p_o is an output parameter of a component $c_o \in C \cup I$, and p_i is an input parameter of component $c_i \in C \cup G$. For example, the link between flight-arrival-date parameter of Reserve-Flight and arrival-date parameter of Rent-Car in Figure 4 below can be represented as <Reserve-Flight, flight-arrival-date, Rent-Car, arrival-date>. When representing parameters and components in examples, we may use their KB names for convenience, as in this case.

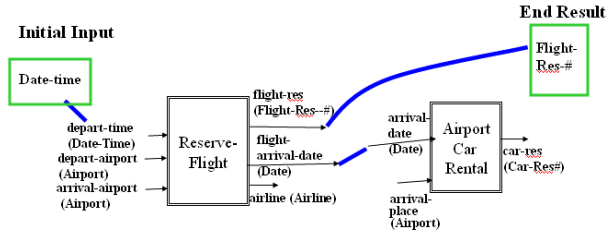

Figure 4. a simple workflow sketch.

Properties of a Workflow

In CAT, the workflow composition process is guided by a set of desirable properties. This section first introduces some features of workflow components and defines those desirable properties in terms of the component features.

Given a workflow <*C, L, I, G*> and its component $c_i \in C$, p ∈ input-parameters(c_i) is *satisfied* iff ∃ a link $<c_o,p_o,c_i,p_i> \in L$ s.t. p_i = p. That is, an input parameter is satisfied when it is linked to any output parameter of a workflow component. Otherwise we call the parameter *unsatisfied*. A workflow component is satisfied if all its input parameters are satisfied. In Figure 4, both Reserve-Flight and Airport-Car-Rental component are unsatisfied.

A Link $<c_1,p1,c_2,p2>$ is *consistent* iff subsumes (range(c1,p1),range(c2,p2)). Otherwise we call it *inconsistent*. In other words, if the link's destination data type (input to) subsumes its source type (output from), then the data being supplied to the destination is guaranteed to be useful.

Given a workflow <*C, L, I, G*> workflow component c1 ∈ *C* ∪ *I*, c1 is *remote-linked* to a workflow component c2 ∈ *C* ∪ *G* iff (∃ link l $<c_o,p_o,c_i,p_i> \in \mathcal{L}$ where c_o = c1 and c_i = c2) or (∃ component c3 ∈ *C* s.t. c1 is remote-linked to c3 and c3 is remote-linked to c2). That is, there exists a (directional) chain of links that connects c1 to c2 in the workflow. In Figure 4, the Initial-Input component

with output parameter Date-Time is Remote-Linked to the End-Result Flight-Res-# via Reserve Flight component.

A Link l $<c_o,p_o,c_i,p_i> \in \mathcal{L}$ is *redundant* iff $\exists$ link l2 $<c_o',p_o',c_i',p_i'> \in \mathcal{L}$ s.t. l $\neq$ l2 and $c_o = c_o'$ and $p_o' = p_o$ and $c_i = c_i'$ and $p_i = p_i'$ or if c_o and c_i are remote-linked.

Given a workflow $<C, L, I, G>$ its component $c \in C$ is *justified* iff $c \in G$ *or* $\exists$ c2 $\in G$ where c is remote-linked to c2. Otherwise, C is *unjustified*. Currently, Airport-Car-Rental component in Figure 4 is unjustified.

The following is a list of desirable properties of workflows, based on the elementary properties listed above.

A workflow W$<C, L, I, G>$ is *purposeful* iff $\exists$ $G \neq \phi$. Otherwise, W is not purposeful. That is, the workflow contains at least one End-Result component. CAT allows users to construct sketches of workflows without specifying desired End-Results, but to complete a workflow, users need to provide the kinds of outcome they expect.

A workflow W$<C, L, I, G>$ is *grounded* iff $\forall$ c $\in C$, executable(c) = true. Otherwise, W is ungrounded. To be able to execute a given workflow, all the components introduced to the workflow should be specialized into executable ones.

A workflow W$<C, L, I, G>$ is *satisfied* iff $\forall$ c $\in C \cup G$, c is satisfied. Otherwise, W is unsatisfied.

A workflow W$<C, L, I, G>$ is *justified* iff $\forall$ c $\in C \cup I$, c is justified. Otherwise, W is unjustified.

A workflow W$<C, L, I, G>$ is *cyclic* iff $\exists$ c $\in C$ s.t c is remote-linked to c. Otherwise, W is acyclic

A workflow W$<C, L, I, G>$ is *consistent* iff $\forall$ link l $\in \mathcal{L}$, l is consistent. Otherwise, W is inconsistent.

A workflow is *correct* if it is purposeful, grounded, acyclic, satisfied, justified, and consistent.

Given a workflow, CAT checks the above properties based on the features of the workflow's components and links, and produces a report on the kinds of problems that made the workflow not correct. The report also includes suggestions for fixing each error.

Workflow Refinement Actions

At any time, the user may do the composition actions below:

- add a component to the workflow. It may be an abstract component, or a specific (executable) one, or an Initial-Input, or an End-Result.
- add a link between two components (or Initial-Input, or End-Result) to indicate that the output of one should be the input of another.

(The user can also delete anything that can be added.)

Action Name	Description	Possible Fixes	Possible New Errors
Add Component(w,c)	Given workflow w = $<C, L, I, G>$ and c $\in$ components(), w becomes $<C \cup \{c\}, L, I, G>$. *Note: addition of an Initial-Input or End-Result is done in a similar way for set I or G, respectively.*	w purposeful	c not grounded or justified, or p $\in$ input-parameters(c) not satisfied
Remove Component(w,c)	Given a workflow w=$<C, L, I, G>$ and c$\in C$, w becomes $<C-\{c\}, L, I, G>$. *Note: removal of an Initial-Input or End-Result is done in a similar way, as with Add actions*. As RemoveComponent simply removes a component without deleting associated links, generating 'dangling' links, we don't allow users to use this primitive action. Instead, users can use RemoveComponentAndLinks, as described below.	c grounded or justified, or p $\in$ input-parameters(c) satisfied	w not purposeful
AddLink (w, c1,p1,c2,p2)	Given a workflow w=$<C, L, I, G>$, c1 $\in C \cup I$, p1$\in$ output-parameters (c1), c2 $\in C \cup G$, and p2$\in$ input-parameters (c2), w becomes $<C, L \cup \{<c1,p1,c2,p2>\}, I, G>$.	p2 satisfied, c1 justified.	New link may not be consistent, cause a cycle, or be redundant with an existing link.
RemoveLink (w, l)	Given a workflow w=$<C, L, I, G>$, l $\in L$, w becomes $< C, L-\{l\}, I, G>$.	l was redundant, not consistent, or cycle-causing.	Unsatisfied parameter, unjustified component.

Figure 5. Primitive actions and possible resulting workflow properties.

Figure 5 details primitive actions in terms of their effects in the workflow (add and remove actions have complementary effects). In addition to these primitive actions, CAT allows "composite" actions, but currently only within suggested error fixes, in order to make the composition process more coherent and efficient. See Figure 7 below for summaries of these actions' effects (in their role as suggested fixes for particular errors). Each

composite action, is a sequence (similar to a macro) of the primitive actions. The current composite actions are:

- AddAndLinkComponent(w, c1, p1, c2, p2): Given a workflow w=<*C, L, I, G*>, c1 ∈components(), p1∈ output-parameters (c1), c2 ∈ *C,* and p2∈ input-parameters (c2), if c1 ∉ *C* do AddComponent(w,c1) and AddLink (w, <c1,p1,c2,p2>). Otherwise just do AddLink (w, <c1,p1,c2,p2>). This action can be used in making unsatisfied parameters satisfied. That is, the action adds a new component (or Initial-Input) and links one of its output parameters to an input parameter of an existing component.
- RemoveComponentAndLinks(w, c): Given a workflow w=<*C, L, I, G*> and component c ∈ *C* ∪ *I* ∪ *G,* ∀ l <c_o,p_o,c_i, p_i> ∈ *L* s.t. c = c_o or c = c_i , do RemoveComponent(w,c) and RemoveLink (w, l). As mentioned in Figure 5, this is a 'cleaner' version of RemoveComponent. It also is currently the only non-primitive action that can be called manually (i.e., outside of a suggested error fix).
- InterposeComponent(w,c1,p1,c2,p2,c',p1',p2'): Given a workflow w=<*C, L, I, G*>, c1 ∈ *C* ∪ *I* , p1∈ output-parameters (c1), c2 ∈ *C* ∪ *G,* p2∈ input-parameters (c2), l <c1,p1,c2,p2> ∈ *L,* c'∈components(), p1'∈ input-parameters (c'), p2' ∈ output-parameters (c'); if c' ∈ *C*, do RemoveLink(w,l), AddLink(w, c1,p1,c',p1'), and AddLink (w, c', p2', c2, p2). Otherwise, do AddComponent (c') first. This action removes an existing link, and links a third component between the old link's output, and to the old link's input. This action is proposed as a suggested fix when there is a component such that at least one of its input parameters subsumes the output parameter of an inconsistent link, and at least one of its output parameters is subsumed by the input parameter of the inconsistent link.
- ReplaceComponent (w,c1,c2): Given a workflow w=<*C, L, I, G*>, c1 ∈ *C,* c2 ∈ components(), do RemoveComponentAndLinks (w,c1) and addComponent(w,c2). This removes an existing component and its links, and adds a different component.

THE ERRORSCAN ALGORITHM

The ErrorScan algorithm references the workflow's desired properties to check for errors in a workflow, and uses the knowledge base to generate fix suggestions for the user. The algorithm is shown in Figure 6. Note that some suggestions represent a list of possible fixes (indicated by "∀"), so there will be one suggestion generated to the user per element of that list.

ErrorScan

Input: Workflow w <*C,L,I,G*>
Output: list of errors and corresponding fix suggestions

I. **If w is not purposeful, return Error.**
  Suggestions: define end result e using types from the KB, **AddEndResult** (w,e).
II. **If w is cyclic, return Error.**
  Suggestions: ∀ links l in the cycle, **RemoveLink**(w,l).
III. For each component c in w:
  a. **If c is not justified, return Error.**
    Suggestions ∀p ∈output-parameters (c), use FindMatchingInput(p) to find components cj in the workflow or the KB s.t. pj ∈ input-parameters(cj) and subsumes(pj,p); **AddLink** (w, c,p,cj,pj)
  b**. If c is not grounded, return Error.**
    Suggestions: ∀ cj ∈ specializations(c), **ReplaceComponent**(w, c, cj).
  c. For each i ∈ input-parameters(c):
    1. If i is not satisfied, return Error.
      Suggestions: ∀ cj ∈ (*C* ∪ *I* ∩ FindMatchingOutput(i)), with pj ∈output-parameters(cj) s.t. subsumes(range(c,i),range(cj,pj)) **AddLink**(w, cj,pj,c,i).
      Suggestions: ∀ cj ∈ FindMatchingOutput (i)) s.t. cj ∉ *C* ∪ *I* **AddAndLinkComponent**(w, cj)
IV. For each Link l in w:
  a. **If l is not consistent, return Error.**
    Suggestions: ∀ ci ∈ FindInterPosingComponent(l), **InterposeComponent** (…).
    Suggestion: **RemoveLink(**l).
  b. **If l is Redundant, return Error.**
    Suggestion: **RemoveLink** (l).

Figure 6: The ErrorScan algorithm.

If ErrorScan does not generate any errors for a given workflow, the workflow is purposeful, acyclic, grounded, satisfied, justified, and consistent, and therefore correct. It is possible to configure different versions of the algorithm to check different subsets of these properties.

ERROR #	SUGGESTED FIX	NEW ERRORS
I: w is not Purposeful	AddEndResult	IIIc1
II. w is cyclic	RemoveLink	IIIc1
IIIa: c is Unjustified	AddLink	
IIIb: c is not Grounded	Specialize Component	IIIa, IIIc1

IIIc1: In c, i is Unsatisfied	AddLink	
	AddAndLink Component	IIIa, IIIb, IIIc1
IVa: l is not Consistent	Interpose Component	IIIa, IIIb, IIIc1
	RemoveLink	IIIa, IIIc1
IVb: l is Redundant	RemoveLink	

Notation: W=Workflow; C=Component; I=Input Parameter; L=Link.

Figure 7: Errors, fix suggestions, and new errors potentially generated by applying that fix.

As the user and system mutually search the space of workflows, with the goal of moving towards the subspace of correct workflows, the fix suggestions generated by ErrorScan always fix the error they were associated with, and generally tend to move the user closer towards a correct workflow. Figure 7 lists the errors potentially generated by applying a given fix. These new errors fall into a subset of the error conditions (IIIa: unjustified component, IIIb: ungrounded component, and IIIc1: unsatisfied input), each of which errors in turn can potentially be found and fixed by ErrorScan.

In the same interest of providing intelligent assistance, the ErrorScan algorithm filters and orders its suggestions. One heuristic is that each suggestion is preferably a sequence of actions that, as a whole, fixes more errors than it causes. For instance, the algorithm prefers not to suggest adding a new component if an appropriate component is already included in the Workflow. Also, ErrorScan will not suggest removing a component until after suggesting all other options, though the user can always remove components if added in error. This level of assistance can be contrasted with unguided manual composition (using the primitive actions from Figure 5), where any unsuggested action can lead to more errors of any type.

The algorithm also incorporates heuristics for ordering errors as they appear in the interface. For example, errors are ordered most recent first (i.e., generated by the most recent user actions). Errors are also ordered by "serious" errors (those that involve end results or modules linked to end results) first, though the "seriousness" criterion has not been formalized beyond that point.

INITIAL INTEGRATION WITH FULLY AUTOMATED PLANNER

CAT is useful for interactive composition; however, it may become tedious to use if a large number of components are needed in the workflow. This is because composition in CAT, though not entirely manual, is still stepwise; the user makes an action, and CAT critiques the results of that action and suggests one-step lookahead-based fixes for single errors. At the other extreme, if we were to convert CAT to full automation in order to eliminate unnecessary user interaction, we might remove the possibility of user preference expression during composition. Full automation would also require the user to explicitly determine the pathway's end results at the beginning of composition (in order to provide the automatic with a goal state); however, users may not have explicit descriptions of the desired end results or goals in mind before composing the pathway.

Our combined approach to this problem uses automated planning techniques, while also incorporating user preferences during composition. In the interest of achieving a balance in user interaction and efficiency, we have begun to enhance CAT into a mixed-initiative workflow composition tool, AutoCAT, implemented by merging CAT with an existing planner, Prodigy [Veloso et al 95].

AutoCAT takes a partial workflow created by the user, removes inconsistent links and unjustified components, and then sends the partial workflow to the planner as a planning problem (the workflow must be purposeful, i.e. include at least one end result as the goal state). The planner returns the shortest set of components that it added to complete the workflow. During planning, if the planner reaches a state where more than one similar component can be added, it will return the most abstract component of the candidate set as a placeholder (e.g., the planner will choose "abstract flight reservation service" instead of choosing between "Travelocity" or "Orbitz", as those two services are considered too similar for the planner to choose between). AutoCAT incorporates and links the components returned from the planner into the workflow, and then presents the completed workflow to the user, along with suggestions for specializing any abstract components.

AutoCAT is convenient for the user, by completing many component/link additions/removals at once. Additionally, AutoCAT keeps the user "in the loop" by allowing user choice for specializing components, and for otherwise changing the pathway manually, both before and after invoking the automatic planner.

RELATED WORK

Many approaches to web service composition address automatically composing the services [Burstein et al 00, McDermott 02, Sycara et al 99]. However, in many contexts, users will want to drive the process, influencing selection and configuration of components. In addition, users may only have high-level or partial descriptions of the desired outcome or the initial state, so it may be hard to apply automatic approaches with explicit goal representations.

Interactive acquisition of planning preferences and constraints enables users to direct automated or semi-automated planners to search for certain kinds of solutions by influencing the decisions they make throughout the

search. KANAL [Kim and Gil 01] finds errors in a process or plan specified by a user and makes suggestions for fixing those errors. Unlike CAT, KANAL does not rely on formal definitions of desirable properties, and in that sense CAT can be seen as a cleaner formalized version of KANAL. In addition, KANAL did not use a description logic and did not exploit general types of components.

PASSAT and similar systems [Myers 97, Myers et al 02, Myers et al 03] take plan sketches provided by a user and interprets and completes 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 96, Myers 00] 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 ErrorScan.

TRAINS [Allen 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 workflows to follow up with the user in terms of possibly incorrect portions of it.

Interactive algorithms to guide the search of solutions have used visualization techniques to explore the tradeoffs among solutions [Anderson et al 00, Blythe 02]. We do not address how to guide the user through the solution space, but in future work we would like to 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 02, Kim and Blythe 03, McCluskey et al 03] are complementary 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.

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 the 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 90, Alterman 86, Litman 94].

CONCLUSIONS AND FUTURE WORK

This paper presents a flexible approach to interactive workflow composition that combines knowledge-based representations 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 workflow. We defined formal properties of workflows and presented the ErrorScan algorithm for guiding users to specify correct workflows.

Recently, we have been working in conjunction with SCEC and with the Grid computing group [Blythe et al 03] in order to translate CAT's workflow specification into a Grid-executable workflow for the SCEC hazard analysis domain. In order to coordinate CAT with the Grid and with SCEC systems, we are mapping CAT's ontology onto the URLs of real-world SCEC services. In the same interest, the SCEC services specified in CAT's workflows are being augmented to include Grid requirement parameters such as runtime and memory size. The result of this collaboration is that the user would interactively compose a workflow in CAT as before, and then the user could send the complete workflow to the Grid for execution.

To make CAT compatible with increasingly complex workflows, we also expect to extend the formalism outlined in this paper to express and control structures like iterative loops, conditional data flow, non-determinism, exception handling, etc., as in [McIlraith and Son 01]. 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.

REFERENCES

[Aler and Borrajo 02] Aler, R. and Borrajo, D. On control knowledge acquisition by exploiting human-computer interaction. *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems (AIPS -02)*, 2002.

[Allen et al 95] Allen, J. et al. The TRAINS Project: A Case Study in Defining a Conversational Planning Agent. *Journal of Experimental and Theoretical AI*, 1995.

[Alterman 86] Alterman, R. An Adaptive Planner. *Proceedings of AAAI-86*, 1986.

[Anderson et al 00] Anderson, D. et al. Human-Guided Simple Search. *Proceedings of AAAI 2000.*

[Benjamins et al 96] Benjamins, V. et al. Constructing Planners Through Problem-Solving Methods. *Proceedings of the Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW'96),* 1996.

[BizTalk 03] Microsoft BizTalk http://www.biztalk.org, 2003.

[Blythe 02] Blythe, J. Acquisition and Visualization of Planning Preferences and Constraints. *Proceedings of AAAI-02,* 2002.

[Blythe et al 03] Blythe, J. et al. The Role of Planning in Grid Computing. *Proceedings of ICAPS-03,* 2003.

[Burstein et al 00] Burstein, M. et al. Derivation of Glue Code for Agent Interoperation. *Proceedings of the Fourth International Conference on Autonomous Agents,* 2000.

[Chen et al 03] Chen, L. et al. Towards a Knowledge-based Approach to Semantic Service Composition. *Proceedings of 2nd International Semantic Web Conference (ISWC2003)*, 2003.

[Chien et al 96] Chien, S., and Mortensen, H. 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, August 1996.

[Devanbu and Litman 92] Devanbu, P, and Litman, D. CLASP - A Plan Representation And Classification Scheme For A Software Information System. *Artificial Intelligence*, 1996.

[Geodise 03] Geodise. http://www.geodise.org/, 2003.

[Griphyn 03] GriPhyN. The GriPhyN Project. http://www.griphyn.org/index.php, 2003.

[Khoros 03] Khoros. http://www.khoral.com, 2003.

[Kim and Blythe 03] Kim, J. and Blythe, J. Supporting Plan Authoring and Analysis. *Proceedings of intelligent user interfaces,* 2003.

[Kim and Gil 01] Kim, J. and Gil, Y. Knowledge Analysis on Process Models. *Proceedings of IJCAI-01*, 2001.

[Kim et al 04] Kim, J., et al. An Intelligent Interface for Web Service Composition. P*roceedings of Intelligent User Interfaces '04*, 2004.

[Lansky et al 95] Lansky, A. et al. The COLLAGE/KHOROS Link: Planning for Image Processing Tasks. *AAAI Spring Symposium on Integrated Planning Applications,* 1995.

[Litman 94] Litman, D. Plan Recognition through Description Logics. *Proceedings of Knowledge Representation and Reasoning-94,* 1994.

[MacGregor 90] Robert MacGregor. The Evolving Technology of Classification-based Knowledge Representation Systems. *Principles of Semantic Networks: Explorations in the Representation of Knowledge,* Morgan-Kaufman, 1990.

[McCluskey et al 03] McCluskey, L. et al. GIPO II: HTN Planning in a Tool-supported Knowledge Engineering Environment. *Proceedings of Int'l Conf. on Automatic Planning and Scheduling (ICAPS'03),* 2003.

[McDermott 02] Mcdermott, D. Estimated-Regression Planning for Interactions with Web Services. *AI planning systems Conference,* 2002.

[McIlraith and Son 01] McIlraith, S. and Son, T. Adapting GOLOG for programming in the semantic web. *Fifth International Symposium on Logical Formalizations of Commonsense Reasoning*, 2001.

[Myers 96] Myers, K. L. Advisable Planning Systems. *Advanced Planning Technology.* AAAI Press, 1996.

[Myers 97] Myers, K. Abductive Completion of Plan Sketches. *Proceedings of AAAI,* 1997.

[Myers 00] Myers, K. Domain Metatheories: Enabling User-Centric Planning. *Proceedings of the AAAI Workshop on Representational Issues for Real-World Planning Systems,* 2000.

[Myers et al 02] K. Myers et al. PASSAT: a user-centric Planning Framework. *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space,* 2002.

[Myers et al 03] Myers, K. et al. A Mixed-Initiative Framework for Robust Plan Sketching. *Proceedings of Int'l Conf. on Automatic Planning and Scheduling (ICAPS'03),* 2003.

[MyGrid 03] myGrid. http://mygrid.man.ac.uk/, 2003.

[SCEC 03] Southern California Earthquake Center. http://www.scec.org/, 2003.

[Sirin et al 03] Sirin, E. et al. Semi-automatic Composition of Web Services Using Semantic Descriptions. *Web Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS2003: 17-24*, 2003.

[SmartDraw 03] SmartDraw. http://www.smartdraw.com/, 2003.

[Sycara et al 99] Sycara, K. et al. Dynamic Service Matchmaking among Agents in Open Information Environments. *ACM SIGMOD Record, Special Issue on Semantic Interoperability in Global Information Systems,* 1999.

[Veloso et al 95] Veloso, M., et al. Integrating Planning and Learning: The PRODIGY Architecture. *Journal of Theoretical and Experimental AI, 7(1),* 1995.

[Weld 99] Weld, D. Recent Advances in AI Planning. *AI Magazine*, 1999.

[Wellman 90] Wellman, M. Formulation of Tradeoffs in Planning Under Uncertainty. PhD thesis, MIT, 1990.