

An Intelligent Assistant for Interactive Workflow Composition

Jihie Kim, Marc Spraragen and Yolanda Gil

University of Southern California/Information Sciences Institute

Marina del Rey, CA 90292 USA

+1 310 448 8769

{jihie,marcs,gil}@isi.edu

ABSTRACT

Complex applications in many areas, including scientific computations and business-related web services, are created from collections of components to form 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 enable users to formulate workflows, but the editors 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 the components selected and their connections; 2) specifying the workflow flexibly, e.g., top-down, bottom-up, from requirements, or from available data; and 3) taking partial or incomplete descriptions of workflows and understanding the steps needed for their completion. We present an approach that combines knowledge bases (that have rich representations of components) together with planning techniques (that can track the relations and constraints among individual steps). We illustrate the approach with an implemented 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.

Categories and Subject Descriptors

H.5.2 [Information Interfaces and Presentations]: User Interfaces – *User interface management systems*; I.2.0 [Computing Methodologies]: Artificial Intelligence

General Terms: Verification, Algorithms, Human Factors

Keywords: Knowledge acquisition, workflow composition, web service composition, workflow editors, interactive planning

1. INTRODUCTION

Composing computational workflows is essential in many areas, including scientific computations and business-related web services. A new kind of science is emerging from the integration of models developed by individual scientists and groups: end-to-end scientific applications that result from the composition of those individual models. Another example is the composition of web services to

create new applications from existing software components (typically as web services) given a customer's needs.

Workflow editors (e.g., [1,2,3]) have been developed to enable users to select components from a library, and to link their inputs and outputs. However, these tools lack the kind of intelligent assistance required in order to:

- **keep track of details to ensure that a correct workflow is formulated:** Manual composition of workflows, as any user-driven process, is a task prone to errors and inconsistencies. As users edit the workflow 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 workflow, whether they follow a top-down or a bottom-up approach. At any point in time, the system should be able to take a partially specified workflow from the user and make suggestions about how to complete it.
- **systematically generate and manage all of the choices throughout the process:** At any point during the workflow composition, there may be many choices to make: add a component (and if so which one), add a link, replace an existing component 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 workflow.

This paper presents an approach to interactive workflow composition that incorporates 1) *knowledge-rich descriptions* of the individual components and their constraints; 2) *a formal algorithmic understanding of partial workflows*, based on AI planning techniques. Using this approach, a system can 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. Using this approach, we have developed CAT (Composition Analysis Tool) and used it in two distinct domains: a scientific application for earthquake simulation, and a simple travel-planning domain to illustrate the approach.

The paper begins by describing our motivations and goals based on a scientific application. We then describe the knowledge bases that we have developed to represent components, and present the algorithm to analyze a partial workflow and help a user complete it. We illustrate the resulting intelligent interaction with a detailed scenario of use.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

IUI'04, January 13–16, 2004, Madeira, Funchal, Portugal.

Copyright 2004 ACM 1-58113-815-6/04/0001...\$5.00.

2. MOTIVATING GOALS AND REQUIREMENTS

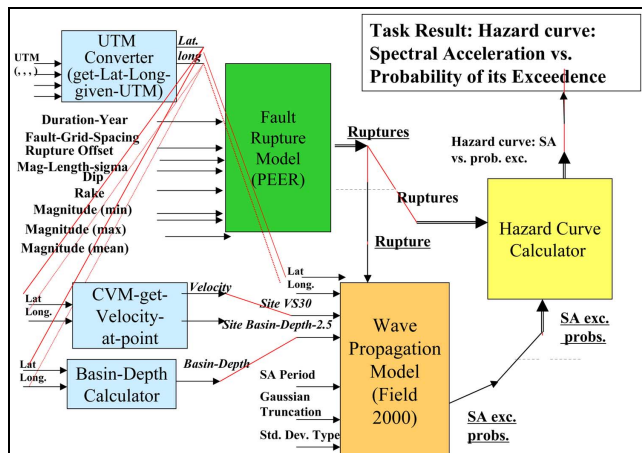


Figure 1. A computational workflow for earthquake simulation analysis.

To motivate the goals of this research we use a concrete scenario from our work with the Southern California Earthquake Center [11], though the basic requirements and problems are shared in other sciences as well [4,10].

Seismic hazard analysis (SHA) 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.

Our ultimate goal is to develop intelligent user interfaces that enable unsophisticated users, such as building engineers and safety officials, to create end-to-end workflows composed of complex scientific models. As a first step to this goal, our work concentrates on assisting users to create a complete and correct workflow, focusing on relatively simple (though still realistic) models and workflows to make the problem more manageable. Currently CAT supports development of valid composition of components (workflow) that can be executed by providing specific data.

Figure 1 shows an example of a workflow. Each component has several inputs; some are provided values directly by the user (e.g., the Duration-Year of the Fault Rupture Model). The user may also choose to have an input take the default value provided by the knowledge base (e.g., the Wave Propagation Model's Standard-Deviation-Type; default = "Total"), and a third group of inputs will take values linked from the results of the execution of other components (e.g., the simulated Rupture data output by the Fault Rupture Model is passed to the input of the Wave Propagation Model).

Engineers may design the workflow in many different ways. One way is to think about it in terms of simulation models. They 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 the particular data they want to look at. Sometimes they want the wave's velocity at the site, or its acceleration. Sometimes they want the probability of earthquake above a certain magnitude affecting that site. Different models provide different types of results. Another way to think about the workflow is the situation they 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.

In summary, users may design workflows using a variety of strategies, including: 1) top-down selection of components, starting from abstract types of models and then replacing them with more specific versions; 2) result-based selection of components, working from desired data to select models that can generate those results; 3) situation-based selection of components, working from the initial data available to select components whose constraints are consistent with those data.

3. APPROACH

Given the goals just described, we are developing a mixed-initiative approach where users can drive the process and the system proactively suggests useful next steps and ensures that the final workflow is correct.

At any time, the user may:

- select a component for inclusion in the workflow. It may be an abstract type of component, or a specific executable one.
- specialize a component to a more specific one
- establish a link between two components to indicate that the output of one should be the input of another.
- specify input data to the overall workflow
- specify desired results

(Users can also delete components and links.)

At any time, the system should analyze the workflow, and suggest possible next actions to the user.

We use two key techniques in our approach:

1) knowledge-based descriptions of simulation models that support reasoning about multiple abstraction hierarchies of components. A given executable simulation model may be categorized in several abstract classes depending on the features being abstracted. In order to reason adequately about an abstract type of model, the system should be able to represent common features that apply to all models of that type. We use description logic to reason about ontologies of component descriptions.

2) analysis of partially constructed workflows based on AI planning techniques. In our current framework, a workflow is a directed acyclic graph: components are nodes and links are directed edges. At any point during composition, the system can search this graph for missing components, for links that are inconsistent with the descriptions of the components, for unnecessary components, etc. Planning algorithms provide a useful framework to relate steps to goals and initial states, and can help us formalize a user's actions in terms of incremental plan generation. The next two sections describe each of these two techniques in turn.

3.1 Supporting Knowledge Base

In order to support the kinds of interactions described above, our knowledge base defines the components that can be used in a workflow and their input and output parameters. The *component ontology* has hierarchies of components that describe abstract-level components as well as specific executable components. The *domain term ontology* defines data types for representing input and output parameters and the constraints associated with them. It also defines how parameters of different components are related with each other. For example, it represents how parameters of abstract components are mapped to parameters of more specific ones.

In developing the domain term ontology, pre-existing definitions and constraints can be exploited whenever they are available, though new terms and their associated constraints may need to be added. The component ontology is built based on the domain term ontology by relating inputs and outputs. For example, when a component is available as a web service and defined in WSDL (Web Service Description Language) [13], we can map its “operations” into component types in the component ontology based on input parameters in the request messages and output parameters in the response messages. Currently these ontologies are built by hand but we are exploring approaches for generating component ontologies semi-automatically from existing descriptions of the components, such as WSDL descriptions.

Figure 2 shows a part of the ontologies that we are using in a travel-planning domain. For example, component Car-Rental represents car rental operations, covering different types of car rental services. It can be specialized based on operation sites (Car-Rental-by-City or Car-Rental-by-Airport), vendors (Car-Rental-Enterprise, Car-Rental-BreezeNet, etc.), or both (Car-Rental-Enterprise-by-City). That is, the ontology contains features that characterize and categorize component subclasses. The parameters of the components are mapped to data types in the domain term ontology, as shown in the figure. For example, Car-Rental-by-City has two input parameters (Arrival-City and Arrival-Date) and one output parameter (Car-Reservation).

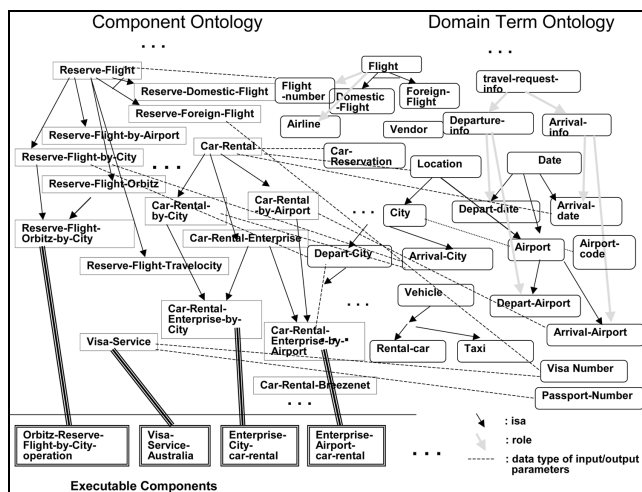


Figure 2. Example Component Ontology and Domain Term Ontology.

Note that 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, as described below.

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 details of errors and suggestions are described in the following section.)

The CAT queries to the knowledge base include:

- Get-Input-Parameters (Component)
- Get-Output-Parameters (Component)
- Subsumes (Data-Type1, Data-Type2)
- Subsumes (Component1, Component2)
- Executable (Component): check whether Component is an actual executable Component and not an abstract type
- Get-specializations (Component[,role, val]): retrieve subconcepts of Component1, optionally only where value for role “role” is “val”.
- Find-Component-with-Output-data-type (Data-Type)
- Find-Component-with-Input-data-type (Data-Type)
- Get-system-default-value (Component, Parameter)

3.2 An Algorithm to Analyze Partial Workflows

The analysis of partial workflows created by the user is done using an AI planning framework [14]. Each component is treated as a step in the plan, the inputs of a component are the preconditions of that step and the outputs are its 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) are akin to a refinement operator in plan generation. User Provided Data (initial data) and End Results (desired data) are handled uniformly as any other component, the former as a component with no input parameters and the latter as a component with no outputs. 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.

We have developed a domain-independent algorithm to support mixed-initiative workflow creation that assists the user by ensuring that the workflow is well-formed and executable. Specifically the final workflow must be compliant with a set of desirable properties:

- **Tasked** — contains one or more End Results
- **Satisfied** — all input parameters of all components are provided by other components, by default values, or as user inputs
- **Grounded** — all components are executable (i.e., not abstract)

- **Justified** — *at least one output parameter of each component is linked to an End Result or to another Justified component*
- **Consistent** — *each link connects an output of a component to the input parameter of another component, where the former subsumes the latter (its type is more general in the ontology)*
- **Unique** — *no link or component is redundant with any other one*

A full formalization of these properties and the algorithms below is provided elsewhere [5].

Note that these properties can also help relate the workflows generated by a user to workflows that an automated approach could generate. Workflows that contain errors of consistency and uniqueness would never be generated automatically. Two additional properties are then useful. A partial workflow is **correct** if it is consistent and unique. A workflow is **complete** if it is satisfied and tasked. Automated approaches always form workflows that are complete, correct, and justified. Our algorithm considers also the space of workflows that do not have these three properties, due to user error and non-systematic exploration of options that are natural in mixed-initiative approaches. Also note that our approach could be combined with an automated approach if our system is configured to help the user create a correct and justified workflow, which then the planner could use it as a starting point to fulfill the other properties and create a complete and grounded workflow.

A workflow contains an error for each item that does not comply with these properties: links may be inconsistent or redundant, components may be unjustified, ungrounded, and the workflow itself may be untasked.

Now the potential effects of the user's primitive actions can be described more fully, in terms of the errors that they may introduce and the properties that they satisfy. Note the symmetry between add and remove actions, in terms of the potential fixes and errors generated.

Add Component

Possible Fixes: Tasked workflow if new component is the only End Result.

Possible New Errors: New component may not be Satisfied, Grounded, or Justified.

Remove component

Possible Fixes: Removed component may have been not Satisfied, Grounded, or Justified.

Possible New Errors: Tasked workflow if removed component was the only End Result, inconsistent links if component had any links.

Add Link

Possible Fixes: Satisfied component if it now has no input parameters without values. Justified component, if now linked to an End Result (or to a component linked to an End Result).

Possible New Errors: New link may not be Consistent or Unique.

Remove Link

Possible Fixes: Removed link may have been not Consistent or Unique.

Possible New Errors: Not Satisfied component if it now has any input parameters without values. Unjustified component if now

not linked to an End Result (or to a component linked to an End Result).

ErrorScan Algorithm

Input: workflow

Output: errors and corresponding suggestions

I. If workflow is not Tasked

Suggestions: add [choose from possible End Results]

II. For each component within workflow

a. If component is not Justified

Suggestions: Remove component and its links, or link to [choose from list of already Justified components].

b. If component is not Grounded

Suggestions: Specialize component to [choose from specialized versions of component].

c. For each input parameter of component

1. If input parameter is not Satisfied

Suggestions: link from [choose from list of workflow and knowledge base components' appropriate output parameters], use [parameter default value] from knowledge base, or enter a value manually.

III. For each link within workflow

If link is not Consistent

Suggestions: Remove link, or fix link by interposing and linking [choose from appropriate components].

If link is not Unique

Suggestions: Remove link.

Figure 3. The ErrorScan Algorithm

We have developed the ErrorScan algorithm to check whether a given workflow is compliant with the properties above. Each deviation from these properties, by the workflow or by one of its elements is reported as an error 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 Figure 3.

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 subsume the type of that parameter).

In the interest of providing intelligent assistance, 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. For instance, the system would never suggest adding an inconsistent link to fix an unsatisfied component. The suggestions tend to be additive or corrective, i.e., the system will not suggest removing a component or link unless added incorrectly by a user not following previous suggestions. In summary, if the user consistently applies suggested fixes, this will help to generate a workflow whose components conform to the desirable properties listed above.

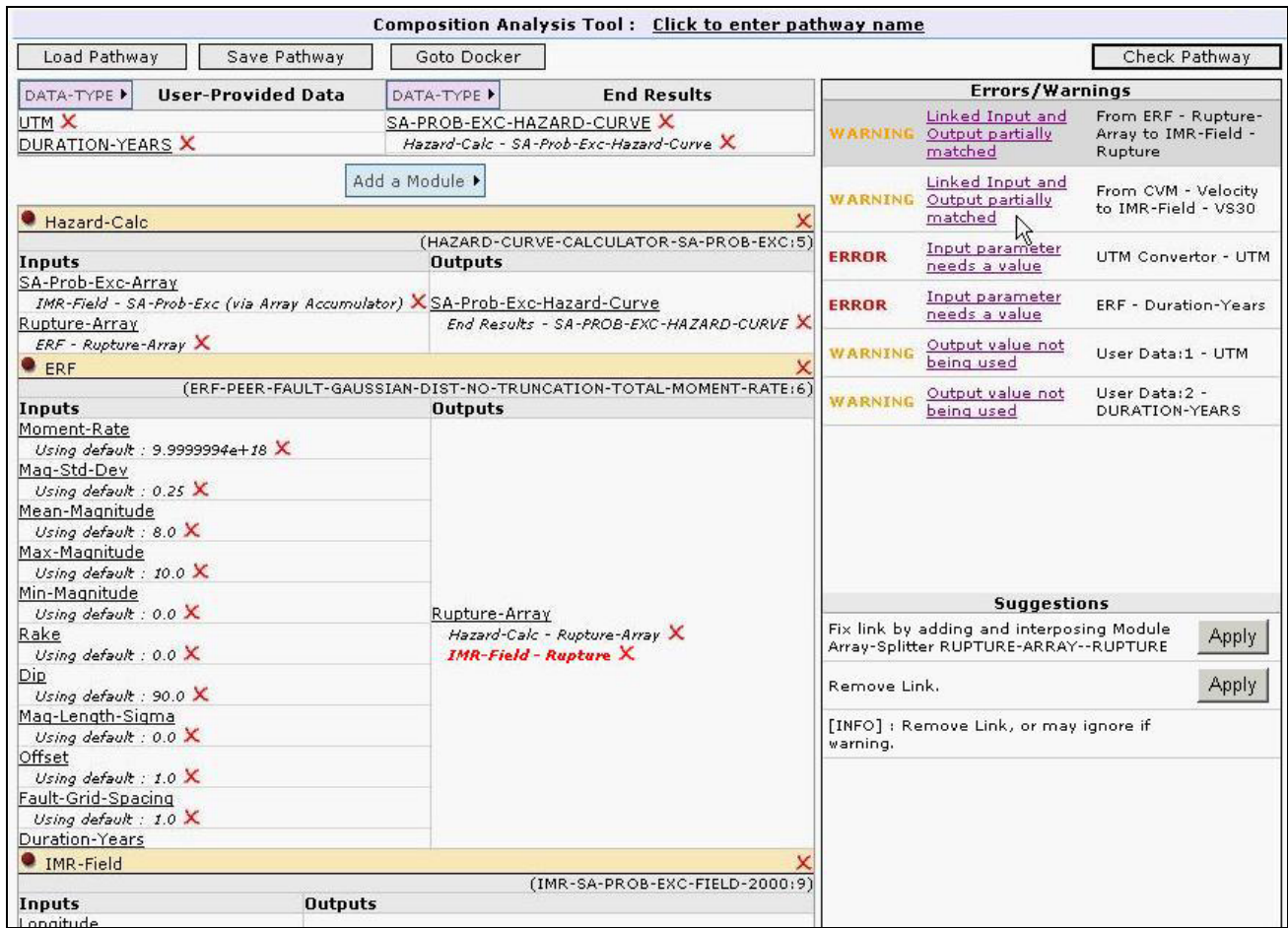


Figure 4. CAT Interface, showing the workflow on the left and the system’s suggestions on the right.

We also incorporated heuristics into the algorithm for ordering errors and suggestions as they appear in the interface. Errors are ordered most recent first (i.e., generated by the most recent user actions), and then by the more serious errors before warnings. For example, a non-unique link would only generate a warning, as a workflow with redundant links may still be correct. Suggestions are ordered with specific fixes listed before informational messages, and the algorithm filters out fixes that involve the addition of new components to the workflow, if similar fixes can be applied using components already in the workflow.

4. EXAMPLE

Figure 4 shows the current web-based interface (for earthquake science domain). On the left of the screen are components and links currently in the workflow. On the right are workflow errors, and suggestions for the selected error. For illustration purposes, the example detailed below will use a simpler travel-planning domain. Also, subsequent screenshot figures will be cut/pasted from the various areas of the interface; each figure will contain elements from the components/links area followed by elements from the errors/suggestions area.

The user wants to find a flight reservation number and, based on the flight details, a car rental reservation at the arrival airport. The user adds “Reserve Flight” component and “Car Rental” component, and links these components’ outputs to the appropriate End Results. Fig. 5 shows the errors that are now present in the workflow. Note that

the added components are both not executable (i.e., they represent generic families of services). Suggestions for specializing “Car Rental” are also shown in Fig. 5, based on the component ontology (i.e., how they are related to each other).

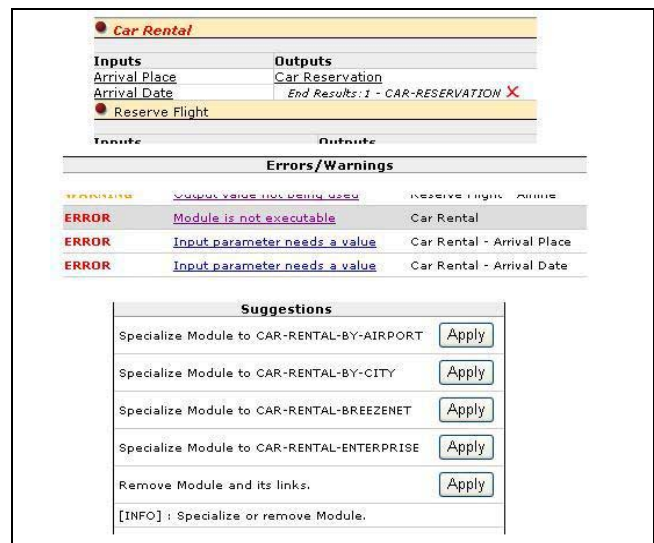


Figure 5. Component added by applying system’s suggestion; system points out that some components are not executable.

This case represents the system’s “top-down” flexibility, in which the user can compose a workflow using abstract components, and when desired, specialize those components as necessary.

The screenshot shows a workflow editor with two main components: 'Car Rental' and 'Reserve Flight'. The 'Car Rental' component has inputs for 'Arrival City' and 'Arrival Date', and an output for 'Car Reservation'. The 'Reserve Flight' component has inputs for 'Arrival City', 'Departure City', 'Departure Date', and 'Arrival Date', and outputs for 'Airline', 'Flight Number', and 'Arrival Date'. A table of errors/warnings is visible, listing several 'Input parameter needs a value' errors and 'Output value not being used' warnings. At the bottom, there are suggestions for linking modules and applying system defaults.

Figure 6. Abstract component specialized by applying system’s suggestion; system finds more unsatisfied inputs.

In Fig. 6, the user has fully specified (by location and vendor) “Car Rental” to “Car-Rental-Enterprise-by-City”. “Reserve Flight” has similarly been fully specialized to “Reserve-Domestic Flight-Orbitz-by-City”. The value of CAT’s mixed-initiative interface is notable here. Instead of having to encode preferences and evaluate multiple, similarly valid plans, CAT allows user preference to resolve unconstrained choices at each step. Note also that the link from the newly specialized “Car Rental: Car-Reservation” parameter, to “End Result 1: Car-Reservation”, remains intact between Figures 5 and 6. In order to provide this automated intelligent assistance, CAT’s component ontology defines how the parameters of different component types are related with each other, and can track parameter equivalencies between parent (less specific) and child (more specific) components.

Now, most of the remaining errors in the workflow are the inputs to the two components, which have no values assigned. However, note that the system suggests that at least one of these inputs, “Car Rental: Arrival Date” can be satisfied by linking the similarly-typed output from “Reserve Flight: Arrival Date” (Figure 6).

In Fig. 7, the user has provided additional data (Departure City, Arrival City, and Departure Date), and linked these to the appropriate input parameters of the “Car Rental” and “Reserve Flight” components, so that no components are not Satisfied. This demonstrates CAT’s flexibility: even when the necessary user data is not provided at the start of the composition, the system can still provide useful suggestions to make progress. The figure represents a correct workflow; the only error remaining is a warning about unused output, which may be ignored.

This screenshot shows the workflow after the user has provided additional data. The 'Car Rental' component now has 'User Data: 2 - CITY' for 'Arrival City' and 'User Data: 1 - CITY' for 'Arrival Date'. The 'Reserve Flight' component has 'User Data: 2 - CITY' for 'Arrival City', 'User Data: 1 - CITY' for 'Departure City', and 'User Data: 3 - DATE' for 'Departure Date'. The error/warnings table now only shows a 'WARNING Output value not being used' for 'Reserve Flight - Airline'. Suggestions are provided to link to existing modules or ignore the warning.

Figure 7. User-provided data is linked to unsatisfied inputs.

The screenshot shows the workflow extended with a new 'Reserve Flight' component (specialized to 'Reserve-Foreign-Flight-Orbitz-by-City') and a 'Visa Service' component (specialized to 'Visa-Service-Australia'). The 'Reserve Flight' component has inputs for 'Visa Number', 'Arrival City', 'Departure City', and 'Departure Date'. The 'Visa Service' component has an input for 'Passport Number'. An error/warnings table shows an 'ERROR Linked Input and Output mismatched' between 'User Data: 4 - PASSPORT-NUMBER' and 'Reserve Flight - Visa Number'. Suggestions are provided to fix the link by adding and interposing the 'VISA-SERVICE' module.

Figure 8. Extending a previous workflow for a new situation.

This screenshot shows the workflow after applying the system's suggestion. The 'Passport Number' input of the 'Visa Service' component is now linked to the 'Visa Number' output of the 'Reserve Flight' component. The error/warnings table now only shows a 'WARNING Output value not being used' for 'Reserve Flight - Airline'. The suggestions section is empty, indicating that the link has been successfully resolved.

Figure 9. Inconsistency fixed by applying system’s suggestion.

Suppose after several times using the same workflow to arrange domestic trips, the user decides to change "Reserve Flight" to book a foreign flight. There is a new wrinkle: foreign flights reservations require a Visa number. The user, realistically, attempts to provide her passport number for this input (Fig. 8), though this action was not suggested by the system. When invoked, ErrorScan reports that the link input and output types are mismatched (inconsistent). However, the system notes that there is a component "Visa Service" which can be interposed into the inconsistent link, creating two correct links (as shown in Fig. 9, wherein the error level is resolved to warnings-only as in Figure 7). This example demonstrates the potential of the system to suggest intelligent solutions to workflow errors. Another point of this scenario is that the system can find fixes for any user errors, although the interface is flexible enough to allow the user to disregard suggestions by manually adding and deleting components and links.

5. RELATED WORK

Graphical tools to lay out a workflow and draw connections among steps abound (e.g., [1,2,3]) but the tools are limited to simple checks on the process models, because there is no semantics associated with the individual steps and links. In contrast, we assume a knowledge-rich environment where the system can check whether the workflow makes sense within the background knowledge that it has.

AI planning approaches for web service composition focus on automatically generating compositions that satisfy user given goals [7,8]. Our work provides an alternative and complementary approach addressing the issues that arise when users want to influence the composition process, including selection of components and their configuration. Our interactive framework also helps when the user has an incomplete description of goals or initial state since he/she can start from high-level task types and then gradually introduce new requirements and preferences while the workflow is being built.

There has been increased interest in mixed-initiative approaches to constructing plans [9, 12] but they are not designed to exploit background knowledge and ontologies, which we believe is crucial technology to providing the strong guidance needed by end users.

6. SUMMARY AND FUTURE WORK

We presented an approach to interactive workflow composition that combines knowledge bases that have rich representations of components together with planning techniques that can track the relations and constraints among individual steps. The tool we have developed has been applied to two different applications: constructing workflows in travel planning domain and constructing computational workflows in earthquake science domain.

Our plans for future work include dynamic generation of component ontologies, by deriving abstract component classes from initial component descriptions, incorporation of automatic composition approaches to our interactive framework, and user evaluations.

7. ACKNOWLEDGEMENTS

We would like to thank researchers at the Southern California Earthquake Center for very valuable discussions. We would also like to thank Varun Ratnakar and Sid Shaw for their help with the CAT system implementation. This research was funded by the National Science Foundation (NSF) with award number EAR-0122464.

8. REFERENCES

1. Chin Jr, G., Leung, L. R., Schuchardt, K., and Gracio, D. New paradigms in problem solving environments for scientific computing. *Proceedings of Intelligent User Interfaces '02*, pp. 39-46, 2002.
2. Edge Diagrammer. <http://www.pacestar.com/edge/>.
3. KHOROS PRO 2001. <http://www.khoral.com/>.
4. GriPhyN. <http://www.griphyn.org/>.
5. Kim, J., Gil, Y., and Spragen, M. A Knowledge-Based Approach to Interactive Workflow Composition. ISI Internal Project Report, 2003.
6. Kim, J. and Gil, Y. User Studies of an Interdependency-Based Interface for Problem-Solving Knowledge. *Proceedings of the Intelligent User Interface*, 2000.
7. McIlraith, S. and Son, T. Adapting Golog for programming in the semantic web. Fifth International Symposium on Logical Formalizations of Commonsense Reasoning, 2001.
8. McDermott, D. Estimated-Regression Planning for Interactions with Web Services. *AI Planning Systems Conference*, 2002.
9. Myers, K., Jarvis, P., Tyson, M., and Wolvertson, M. A Mixed-Initiative Framework for Robust Plan Sketching. *Proceedings of Int'l Conf. on Automatic Planning and Scheduling*, 2003.
10. NVO (US National Virtual Observatory). <http://www.us-vo.org/>.
11. SCEC (Southern California Earthquake Center). <http://www.scec.org/>.
12. Smith, S., Lassila, O., and Becker, M. Configurable mixed initiative systems for planning and scheduling. In A. Tate, editor, *Advanced Planning Technology*. AAAI Press, 1996.
13. W3C: WSDL specification. <http://www.w3c.org/TR/WSDL/>.
14. Weld, D. Recent Advances in AI Planning. *AI Magazine*, 1999.