Accepted by Intelligent User Interfaces Conference 2004

An Intelligent Assistant for Interactive Workflow Composition

Jihie Kim, Marc Sparagen 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 computational workflows. In many cases end users have requirements and preferences that often depend on how the workflow unfolds and cannot be specified beforehand. Workflow editors enable users to formulate workflows but they 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) 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.

Keywords
Knowledge acquisition, workflow composition, knowledge-based approaches

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, to result in 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 out from existing software components (such as software modules or web services) given a customer's needs.

Workflow editors have been developed to support scientific and business applications among others [3,4,5,13,15]. These tools enable users to select components from a library and to specify link their inputs and outputs. However, these tools lack the kind of intelligent assistance required:

- **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 related 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 travel planning domain that is more accessible and helps 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 describe 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.

MOTIVATING GOALS AND REQUIERMENTS

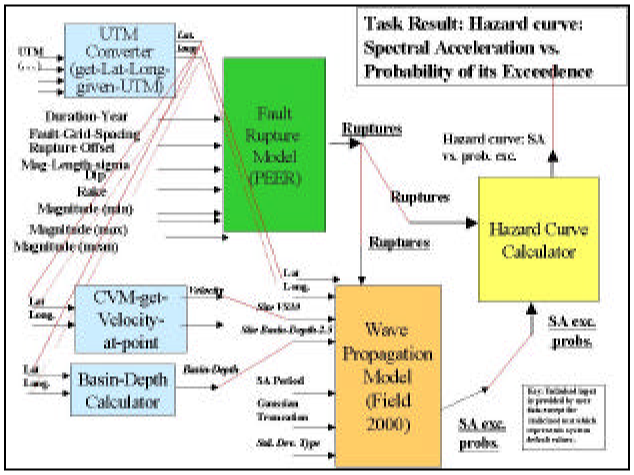


Figure 1. A computational workflow for earthquake simulation analysis.

To motivate the goals of this research we use a concrete scenario from our work with the Southern California Earthquake Center [18], though the basic requirements and problems are shared in other sciences as well [7,15].
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 such 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 Ruptures to be simulated by the Fault Rupture Model are 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 physic-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 the soil type at this site, distance to existing faults, etc), and then project these characteristics to select among the models available.
In summary, users may design workflows using a variety of strategics, including: 1) top-down selection of components, starting from abstract types of models and then selecting specific ones; 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.

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. At any point during the interaction, the system should detect missing components, links that are inconsistent with the descriptions of the components, 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.

Supporting Knowledge Base

In order to support the kinds of interactions described above, our KB defines the components that can be used in a workflow and their input and output parameters. The *component ontology* has hierarchies of components that describes 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 an abstract component are mapped to parameters of more specific ones.

In developing these two ontologies for a given domain, we take existing description of workflow components and map their input and output data types to objects and roles in the domain. Pre-existing definitions and constraints can be exploited whenever they are available. New terms and their associated constraints may need to be added to the domain term ontology. The component ontology is built based on the domain term ontology by relating their inputs and outputs. For example, when a component is available as a web service and defined in WSDL (web service description language) (WSDL 2003), we can map its "operations" into component types in the component ontology based on their input parameters in the request messages and output parameters in the response messages. Currently these ontologies are built by hand but we are exploring new approaches for generating component ontologies semi-automatically from existing descriptions of the components, such as WSDL descriptions.

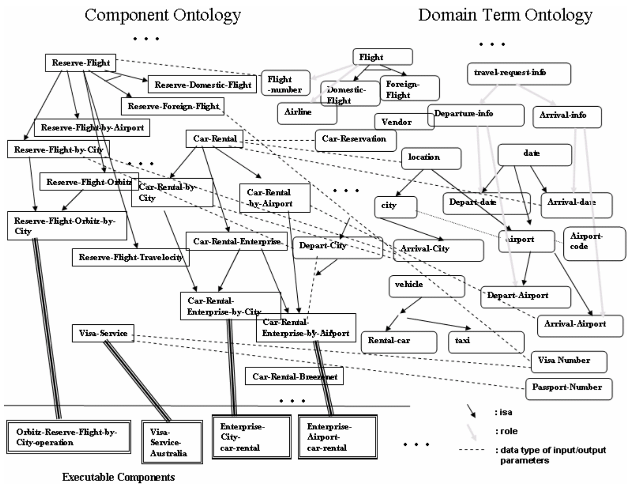


Figure 2. Example Component Ontology and Domain Term Ontology.

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 different subclasses, and executable components are categorized based on such features. 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 have two input parameters (Arrival-City and Arrival-Date) and one output parameter (Car-Reservation).

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. 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 in the example section.
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 KB include:

- Get-input-parameters (Component1)
- Get-output-parameters (Component1, output-params)
- Subsumes (Data-Type1, Data-Type2)
- Subsumes (Component1, Component2)
- Executable (Component1): check whether Component1 is an actual executable component
- Get-specializations (Component1): retrieve subconcepts of Component1
- Get-specializations (Component1, role1, val1): retrieve subconcepts of Component1 whose value for role role1 is val1.
- Find-component-that-produces-data-type (Data-Type)
- Find-component-that-takes-data-type (Data-Type)

An Algorithm to Analyze Partial Workflows

The analysis of partial workflows created by the user is done using an AI planning framework [20]. 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. Input data and end results 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 an 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 used by another component or is an end result*
- **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 [17].

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 actions can be described more fully, in terms of the possible errors that they introduce and the properties that they satisfy:

<u>Add Component</u>
Possible Errors: Tasked Workflow if new component is the only End Result.
Possible New Errors: New component may not be Satisfied, Grounded, or Justified.
<u>Remove component</u>
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).

Note the symmetry between add and remove actions, in terms of the potential fixes and errors generated.

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. The algorithm is shown below.

ErrorScan Algorithm

```
Input: partial workflow W
Output: list of errors and corresponding suggestions

I. If Workflow is not Tasked…
  Suggestions: add End Result [list of possible End Results]
II. For each Component within Workflow (including End Results )…
    a. If Component is not Justified (i.e., Component is not
       an End Result, not linked to an End Result, and
       not linked to a Component that is linked to an End Result) …
       Suggestions: Remove Component and its associated Links, or link
       Component to an End Result or to a Component linked to an End
       Result [list of appropriate Components].
    b. If Component is not Grounded (eg, generic "Flight Finder")…
       Suggestions: Specialize Component [list of specialized versions of Component].
    c. For each Input Parameter of Component…
          1. If Input Parameter is not Satisfied…
             Suggestions: Link from another Component's
             Output Parameter [list of appropriate
             Components / Output
             Parameters], use system Default Value [Default
             Value], or enter a value manually.
III. For each Link within Workflow…
    a. If Link is not Consistent (Input and Output
       Parameter types are mismatched or null)…
       Suggestions: Remove Link, or fix Link by
       interposing and linking a Component [list of
       appropriate Components].
    b. If Link is not Unique (has same Input and
       Output Parameters as another Link)
       Suggestions: Remove Link or ignore.
```

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 the suggestions (e.g., if an input parameter is not satisfied it will get 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 the suggested fixes, this will help to generate a workflow whose components conform to the 6 desirable properties listed above.

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 (error 6 above) 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.

EXAMPLE

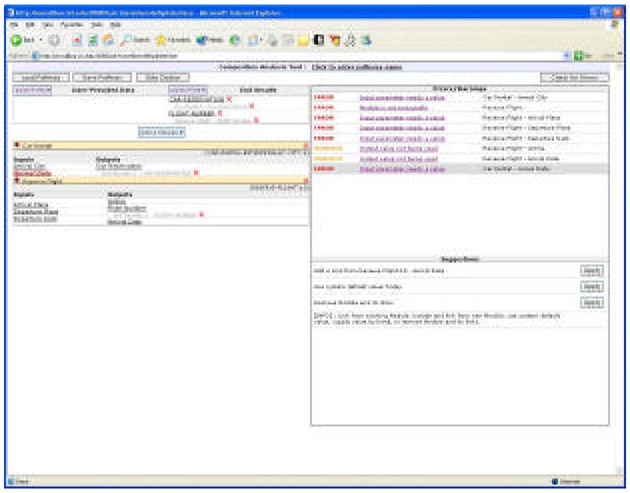
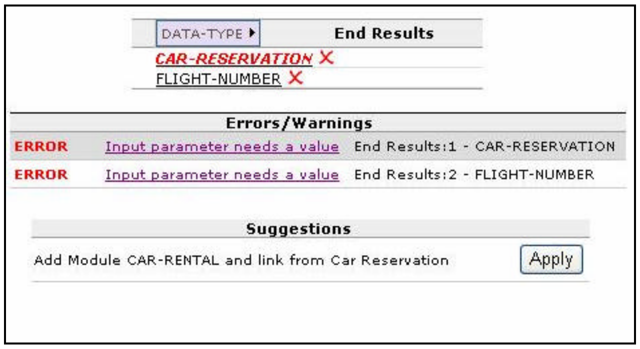


Figure 3.1: Example-- CAT Interface view



DATA-TYPE	End Results
	CAR-RESERVATION ✗
	FLIGHT-NUMBER ✗

Errors/Warnings

ERROR	Input parameter needs a value	End Results:1 - CAR-RESERVATION
ERROR	Input parameter needs a value	End Results:2 - FLIGHT-NUMBER

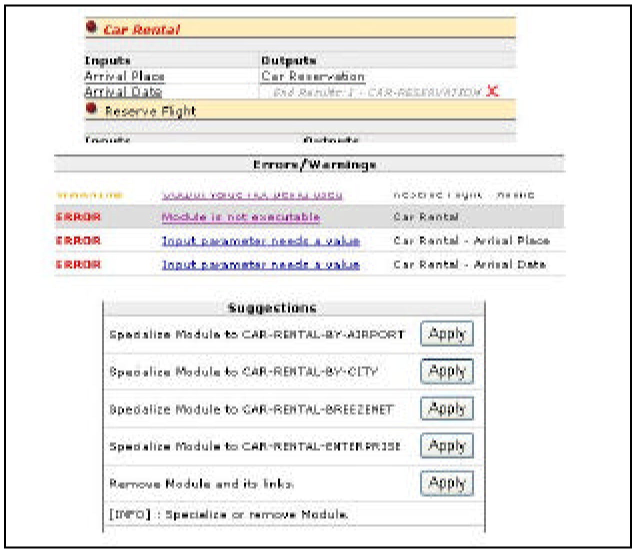
Suggestions

Add Module CAR-RENTAL and link from Car Reservation	Apply

Figure 3.2: Example-- Beginning composition (pieces of interface cut/pasted to conserve space)

Figure 3.1 shows the current CAT interface where the user composing a flight reservation and car rental plans from a domain of web services related to travel. The user wants to find a flight reservation number and, based on the flight details, a car rental reservation at the arrival airport. Fig. 3.2 shows a typical starting point: the user only knows the desired End Results (though the system is flexible enough to allow End Results to be added or removed at any time), and asks the system for help. When the user doesn’t have a concrete description of End Results, abstract ones (such as some transportation arrangement) can be also used in the beginning.

In this case, the ErrorScan algorithm reports that the End Results need values (in algorithmic terms, each End Result behaves as an Executable Component with no Output Parameters and one Input Parameter). The system’s fix suggestions are derived from finding Components, with appropriate Output, available in the KB: add/link “Reserve Flight” Component, add/link “Car Rental” Component.



Car Rental

Inputs	Outputs
Arrival Place	Car Reservation
Arrival Date	End Results:1 - CAR-RESERVATION ✗

Reserve Flight

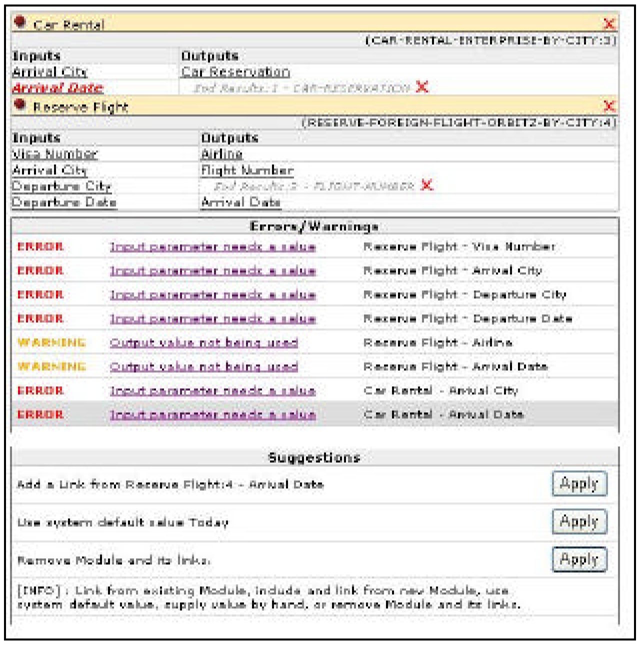
Errors/Warnings

WARNING	[illegible]	[illegible]
ERROR	Module is not executable	Car Rental
ERROR	Input parameter needs a value	Car Rental - Arrival Place
ERROR	Input parameter needs a value	Car Rental - Arrival Date

Suggestions

Specialize Module to CAR-RENTAL-BY-AIRPORT	Apply
Specialize Module to CAR-RENTAL-BY-CITY	Apply
Specialize Module to CAR-RENTAL-BREEZENET	Apply
Specialize Module to CAR-RENTAL-ENTERPRISE	Apply
Remove Module and its links.	Apply
[INFO] : Specialize or remove Module.	

Figure 4.1: Example-- Adding Components by applying suggestions



Car Rental ✗
(CAR-RENTAL-ENTERPRISE-BY-CITY:3)

Inputs	Outputs
Arrival City	Car Reservation
Arrival Date	End Results:1 - CAR-RESERVATION ✗

Reserve Flight ✗
(RESERVE-FOREIGN-FLIGHT-ORBITZ-BY-CITY:4)

Inputs	Outputs
Visa Number	Airline
Arrival City	Flight Number
Departure City	End Results:2 - FLIGHT-NUMBER ✗
Departure Date	Arrival Date

Errors/Warnings

ERROR	Input parameter needs a value	Reserve Flight - Visa Number
ERROR	Input parameter needs a value	Reserve Flight - Arrival City
ERROR	Input parameter needs a value	Reserve Flight - Departure City
ERROR	Input parameter needs a value	Reserve Flight - Departure Date
WARNING	Output value not being used	Reserve Flight - Airline
WARNING	Output value not being used	Reserve Flight - Arrival Date
ERROR	Input parameter needs a value	Car Rental - Arrival City
ERROR	Input parameter needs a value	Car Rental - Arrival Date

Suggestions

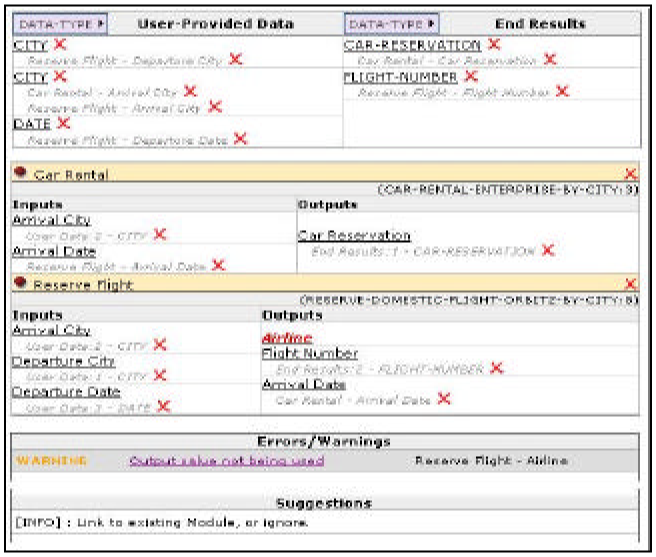
Add a Link from Reserve Flight:4 - Arrival Date	Apply
Use system default value Today	Apply
Remove Module and its links.	Apply
[INFO] : Link from existing Module, include and link from new Module, use system default value, supply value by hand, or remove Module and its links.	

Figure 4.2: Example-- Specializing Components by applying suggestions

The user adds the two Components, as suggested: “Car Rental” and “Reserve Flight”. The system automatically links these Components’ Outputs to the appropriate End Results. Fig. 4.1 shows the errors that are now present in the workflow. Particularly note that the added Components are both not executable (i.e., they do not represent particular real-world executable functions, but rather generic families of services). Suggestions for specializing “Car Rental” are also shown in Fig. 4.1, based on the component ontology (i.e., how they are related to each other). 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.

In Fig. 4.2, the user has fully specified (by location and vendor) "Car Rental" to "Rental Car Reservation by City--Enterprise.com". "Reserve Flight" has similarly been fully specialized to "Reserve Domestic Flight by City-- Orbitz". 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 4.1 and 4.2. 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 4.2).


DATA-TYPE ▸ User-Provided Data
CITY
Reserve Flight - Departure City
CITY
Car Rental - Arrival City
Reserve Flight - Arrival City
DATE
Reserve Flight - Departure Date

DATA-TYPE ▸ End Results
CAR-RESERVATION
Car Rental - Car Reservation
FLIGHT-NUMBER
Reserve Flight - Flight Number

Car Rental
(CAR-RENTAL-ENTERPRISE-BY-CITY:3)
Inputs
Arrival City
User Data:2 - CITY
Arrival Date
Reserve Flight - Arrival Date
Outputs
Car Reservation
End Results:1 - CAR-RESERVATION

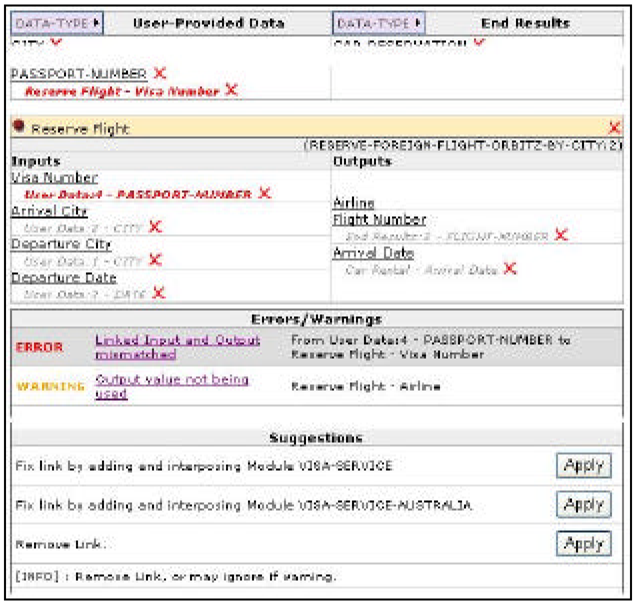
Reserve Flight
(RESERVE-DOMESTIC-FLIGHT-ORBITZ-BY-CITY:8)
Inputs
Arrival City
User Data:2 - CITY
Departure City
User Data:1 - CITY
Departure Date
User Data:3 - DATE
Outputs
Airline
Flight Number
End Results:2 - FLIGHT-NUMBER
Arrival Date
Car Rental - Arrival Date

Errors/Warnings
WARNING Output value not being used Reserve Flight - Airline

Suggestions
[INFO] : Link to existing Module, or ignore.

Figure 5. Example-- Assigning user-provided data to inputs

In Fig. 5, 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 an advantage of 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 pathway, in that the only error remaining is a Warning about unused output, which may be ignored.


DATA-TYPE ▸ User-Provided Data
CITY
PASSPORT-NUMBER
Reserve Flight - Visa Number

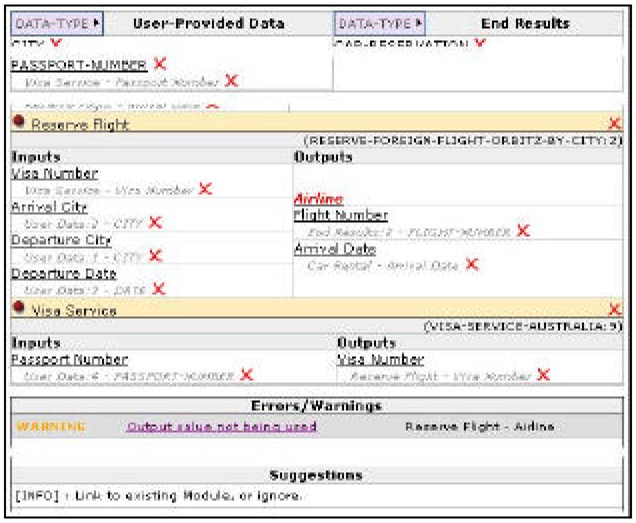
DATA-TYPE ▸ End Results
CAR-RESERVATION

Reserve Flight
(RESERVE-FOREIGN-FLIGHT-ORBITZ-BY-CITY:2)
Inputs
Visa Number
User Data:4 - PASSPORT-NUMBER
Arrival City
User Data:2 - CITY
Departure City
User Data:1 - CITY
Departure Date
User Data:3 - DATE
Outputs
Airline
Flight Number
End Results:2 - FLIGHT-NUMBER
Arrival Date
Car Rental - Arrival Date

Errors/Warnings
ERROR Linked Input and Output mismatched From User Data:4 - PASSPORT-NUMBER to Reserve Flight - Visa Number
WARNING Output value not being used Reserve Flight - Airline

Suggestions
Fix link by adding and interposing Module VISA-SERVICE Apply
Fix link by adding and interposing Module VISA-SERVICE-AUSTRALIA Apply
Remove Link: Apply
[INFO] : Remove Link, or may ignore if warning.

Figure 6.1. Example-- Non-suggested user action may cause complication and inconsistency


DATA-TYPE ▸ User-Provided Data
CITY
PASSPORT-NUMBER
Visa Service - Passport Number

DATA-TYPE ▸ End Results
CAR-RESERVATION

Reserve Flight
(RESERVE-FOREIGN-FLIGHT-ORBITZ-BY-CITY:2)
Inputs
Visa Number
Visa Service - Visa Number
Arrival City
User Data:2 - CITY
Departure City
User Data:1 - CITY
Departure Date
User Data:3 - DATE
Outputs
Airline
Flight Number
End Results:2 - FLIGHT-NUMBER
Arrival Date
Car Rental - Arrival Date

Visa Service
(VISA-SERVICE-AUSTRALIA:5)
Inputs
Passport Number
User Data:4 - PASSPORT-NUMBER
Outputs
Visa Number
Reserve Flight - Visa Number

Errors/Warnings
WARNING Output value not being used Reserve Flight - Airline

Suggestions
[INFO] : Link to existing Module, or ignore.

Figure 6.2. Example-- Inconsistency resolved by 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. 6.1), 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. 6.2, wherein the error level is resolved to warnings-only as in Figure 5). This example demonstrates the potential of the system to suggest complex/intelligent solutions to workflow errors (as opposed to only suggesting to delete the inconsistent Link, which would also be acceptable). Another point of this scenario is that although the interface is flexible enough to allow the user to disregard suggestions by manually adding and deleting Components and Links, the system can always find fixes for any resulting scenario, and those fixes will always guide the user to compose a workflow that satisfies the 6 desired properties as listed in the algorithm section.

RELATED WORK

Graphical tools to lay out a workflow and draw connections among steps abound [3,4,5,13,15] but the tools are limited to simple checks on the process models because there is no semantics associated to 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 [8,9]. 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. Its 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 [11,16] but they are not designed to exploit background knowledge and ontologies, which we believe is crucial technology to providing strong guidance needed by end users.

SUMMAY 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 pathways in earthquake science domain.

Our plans for future work include dynamic generation of component ontologies from existing software components (such as web services), incorporation of automatic composition approaches to our interactive framework, and user evaluations.

REFERENCES

1. Ankolekar, A., Burnstein, M., Hobbs, J.., Lassila, O., Martin, D.,McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Payne, T., and Sycara, K. DAML-S: Web Service Description for the Semantic Web. *ISWC' 02*.
2. The CAT (Composition Analysis Tool) System. http://excalibur.isi.edu:8080/cat/servlet/pInterface.
3. Chin Jr, G., Leung, R., Schuchardt, K., Gracio, D.: New paradigms in problem solving environments for scientific computing. *Proceedings of IUI' 02*, pp. 39-46, 2002.
4. Edge Diagrammer. http://www.pacestar.com/edge/.
5. KHOROS PRO 2001. http://www.khoral.com/ .
6. GriPhyN. http://www.griphyn.org/.
7. Layna Fischer (eds) The Workflow Handbook 2003, 2003.
8. McIlraith, S. and Son, T. Adapting golog for programming in the semantic web. Fifth International Symposium on Logical Formalizations of Commonsense Reasoning, 2001.
9. McDermott, D. and Burstein, M. Extending an estimated-regression planner for multi-agent planning. *AAAI Workshop on Planning by and for Multi-Agent Systems*, 2002.
10. MonachGraph. http://www.singleton-labs.com/mgraph.html
11. Myers, K., Planning with Conflicting Advice. *In proceedings of AIPS' 00*, 2000.
12. NVO (US National Virtual Observatory). http://www.us-vo.org/.
13. Procedure Charter http://www.imagespro.com/programs/2118/.
14. SCEC (Southern California Earthquake Center). http://www.scec.org/.
15. SmartDraw. http://www.smartdraw.com/ .
16. 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.
17. Spraragen, M. Formal representation of CAT properties and algorithms. ISI Internal Project Report, 2003.
18. Sycara, K., Lu, Klusch, M., and Widoff S. Dynamic Service Matchmaking among Agents in Open Information Environments.Journal ACM SIGMOD Record, Special Issue on Semantic Interoperability in Global Information Systems, A. Ouksel, A. Sheth (Eds.), (1999).
19. W3C: WSDL specification. http://www.w3c.org/TR/WSDL/.

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