WISE: a Workflow Information Search Engine

Qihong Shao, Peng Sun, Yi Chen

Department of Computer Science and Engineering
Arizona State University P.O. Box 878809 Tempe, AZ 85287
{qihong.shao, peng.sun, yi}@asu.edu

***Abstract*— Workflows are widely used for representing business processes, web services, scientific experiments, and activities in daily life, like recipes. There is an increasing need for people to search a workflow repository using keywords and retrieve the relevant ones according to their interests. A workflow hierarchy is a three dimensional object containing multi-resolution abstraction views on the same workflow. This unique structure poses a new set of challenges compared to keyword search on tree or graph structures which are typically found in XML or relational data. In this demonstration, we present an effective workflow search engine, WISE, which returns informative and concise search results, defined as the minimal views of the most specific workflow hierarchies containing matching keywords.**

I. Introduction

With the advance of technology, the number of workflows is dramatically growing in scientific and business domains. As a result, workflows are attracting more and more research interest in the database community [3], [1].

A workflow hierarchy provides multi-resolution views of a workflow. For example, Fig. 1(a) shows a workflow hierarchy describing the recipe of curry chicken. A node in a workflow represents a task, which can be a web service invocation, a database query, a program run, an experiment step, or a step in a recipe, etc. A directed solid edge between nodes represents the dependency and dataflow among tasks, referred as *dataflow edge*. For instance, in layer 1 of the recipe, after tasks `make chicken broth` and `preprocess chicken`, their outputs are fed into the next task `cook chicken`. Then the cooked chicken is `served` with `rice pilaf`. A task in the hierarchy can be atomic or composite. A composite task can be expanded/zoomed-in to reveal the detailed procedure of how to perform the task, which is represented by a directed graph consisting of a set of tasks and their dataflows shown in the layer immediately beneath. In Fig. 1(a), each composite task is connected to its expansion by two dotted lines, referred as *expansion edge*s. A composite task `make chicken broth` (0.0), for example, is expanded into a graph as pictured in the leftmost box in layer 2 which consists of three tasks. An atomic task is represented as the leaf nodes in the workflow hierarchy. We also define parent/child relationship on workflow hierarchies in a similar way as the ones in trees [1].

It is very helpful if a user can search relevant workflows in a workflow repository using keywords, and then re-use, include or revise them as needed when designing new workflows, so that the design phase will be easier and be shortened comparing with designing new ones from scratch. Suppose that a user would like to make a dish using chicken breast and coconut milk by sauting, but doesn't have a recipe in mind. She would issue a keyword query Q, "*chicken breast, coconut milk, saute*" on a recipe repository to search for useful ones.

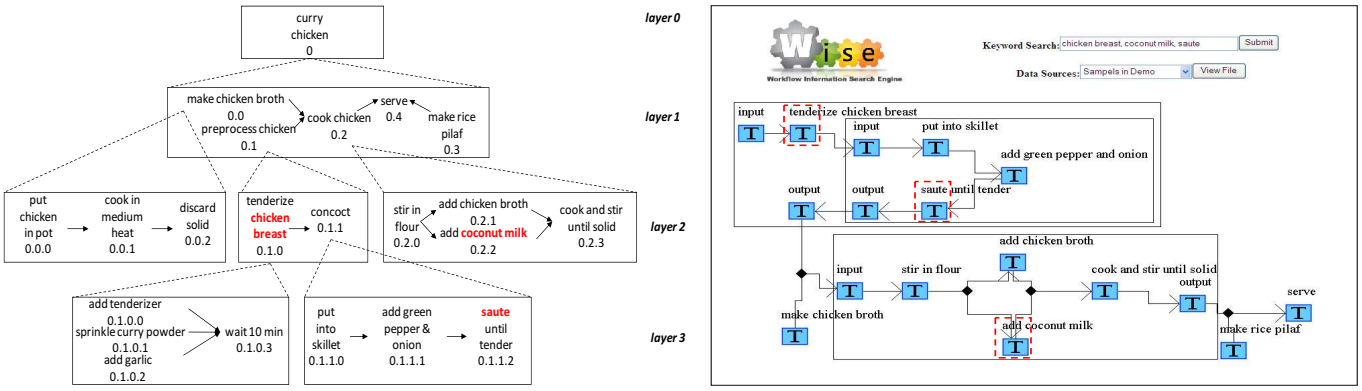
We can easily find the workflow hierarchies in the repository that contain matches to query keywords. Suppose Fig. 1(a) is one of such workflows in the repository, where keyword matches are in red. Obviously, returning the whole workflow hierarchy consisting of nested graphs as a query result is lack of *conciseness* and hard to read, as too much irrelevant information is exposed to the user.

The immediate question is how to define search results when users issue keyword queries on a repository of workflow hierarchies. Recall that much research has been done on keyword search on graph-structured data (e.g., relational databases) or tree-structured data (e.g., XML), where a result is defined as a smallest data tree that contains query keywords. According to this definition, the result of processing Q on Fig. 1(a) will express the relationship of tasks containing query keywords `saute` (0.1.1.2) and `coconut milk` (0.2.2) using a path containing both dataflow edges and across-layer expansion edges: `saute until tender` (0.1.1.2) -> `concoct` (0.1.1) -> `preprocess chicken` (0.1) -> `cook chicken` (0.2) -> `stir in flour` (0.2.0) -> `add coconut milk` (0.2.2).

Unfortunately, such a query result definition is not appropriate for workflow search for two reasons. First, the results fail to be *concise*. The expansion relationships among tasks are included in a query result, which are unlikely to be relevant to the user. Indeed expansion edges are used to define *multiple possible* views of a workflow which have different granularity of abstraction. On the other hand, a user query indicates a *particular* view of the workflow that s/he is interested in, with other views considered as irrelevant. Secondly, the results are not *informative*, as they can not guarantee to capture the dependencies and dataflows among tasks that contain keyword matches. For instance, the relationship among keywords `saute` (0.1.1.2) and `coconut milk` (0.2.2) is obscure.

In our example, it is more desirable to present Fig. 1(b) as a query result, which is a *query-driven view* that visualizes keyword matches shown in red dashed rectangles and their dataflows. For example, after `saute (the chicken) until tender`, we `stir (it) in flour` and then

[1]Typically the dataflow edges between atomic tasks are explicitly recorded in a workflow hierarchy. Other dataflows need to be derived based on these dataflow edges as well as the expansion edges. Fig. 1(a) is presented in the current format to avoid clutter in presentation.



(a) Recipe Workflow Hierarchy

(b) Query Result for Q:{chicken breast, coconut milk, saute}

Fig. 1. Sample Workflow and Query Result of WISE

`add coconut milk`. Note that the dataflow paths among these tasks are not explicitly shown in the workflow hierarchy in Fig. 1(a), but are derived. Also, expansion edges that represent irrelevant views are avoided.

As we can see, supporting keyword search on workflow hierarchies poses new challenges compared with keyword search on relational and XML data. First we need to define meaningful query results. We propose that a query result should be a smallest query-driven view of the workflow hierarchy that contains all keyword matches, which is a graph containing all matches and dataflow edges among them. Second, we must design efficient techniques to generate such query results.

In this demonstration, we present WISE, a Workflow Information Search Engine, which allows users to search in a repository of workflow hierarchies using simple keywords, and returns concise and informative query results. The screen shot of the query result provided by WISE for the sample query is shown in Fig. 1(b).

Our technical contributions include:

1) To the best of our knowledge, this is the first work that supports keyword search on repositories of workflow hierarchies and returns query results capturing the dataflows among tasks matching keywords.
2) We define keyword search results of hierarchical workflows as the *minimal views* of the most specific workflow hierarchies that contain tasks matching keywords, as presented in Section II.
3) We have developed WISE, available at http://wise.asu.edu/, which efficiently and dynamically synthesizes query results by exploiting indexes and labeling schemes, discussed in Section III.

In this paper, we will use a simple recipe workflow hierarchy as our running example. The technique that we propose is applicable for all workflow hierarchies which are composed of hierarchical nested graphs. Such a multi-resolution data structure is a new generalization of graphs and trees, and is widely used in diverse domains, such as business processes, scientific experiments, web services, spatial and temporal data, hierarchical plans, etc.

II. Workflow Retrieval

A. Query Result Definition

In this section we define query results for keyword search on workflows that are *informative* and *concise*. We consider the AND semantics of keyword search, which requires each query result to contain at least one match to each keyword in the query.

We first find workflow hierarchies in the repository that contain matches to all keywords, referred as *relevant workflow hierarchy*. For example, consider our running example Q, `curry chicken` workflow hierarchy contains matches: 0.1.0 to `chicken breast`, 0.2.2 to `coconut milk` and 0.1.1.2 to `saute`, and therefore it is a relevant workflow hierarchy in the repository.

However, returning the whole relevant workflow hierarchy to users is undesirable due to an excessive amount of unnecessary information about irrelevant views and the absence of dataflows among keyword matches.

To define a query result, we first define the *view* of a workflow hierarchy, which can be understood as a projection of a three dimensional workflow hierarchy to a two dimensional plane, flattening out the nested abstraction hierarchy, and visualizing the dataflows among selected nodes. A view V is defined by three conditions with respect to atomic tasks (leaf nodes) and their dataflows in the workflow hierarchy H. (1) A view can only have dataflow edges, but not expansion edges; (2) the node set of a view covers all the leaf nodes in H: every leaf node must have one of its ancestors or itself appear in V; (3) there is a dataflow edge from node u to v in view V if and only if each of them has a descendant-or-self leaf node, u' and v', respectively, such that there is a dataflow edge from u' to v' in H.

Obviously a workflow hierarchy can have many views, as a three dimensional object can have many projections depending on the viewing planes. For example, the five nodes in layer 1 in Fig. 1 (a) compose a view of the curry chicken

workflow hierarchy; Fig. 1(b) is another view of curry chicken, consisting of nodes from different layers. On the other hand, layer 2 in Fig. 1(a) is not a view of this workflow hierarchy, as it does not contain any ancestor-or-self of leaf task `serve` (0.4), violating condition (2); and misses dataflows, violating condition (3).

As we can see, the notion of a view effectively projects a three dimensional workflow hierarchy into a two dimensional plane and preserves the dataflows among the tasks in the view. Thus a natural way of defining keyword search results on workflows would be to return a view of each relevant workflow hierarchy that contains all query keywords.

However, a relevant workflow hierarchy generally has multiple views that contain all query keywords. For example, Fig. 1(b) is a view of curry chicken, containing all keywords in Q. If we choose to replace `make chicken broth` (0.0) with its three children, we get another view of curry chicken that contains all keywords. In order for a query result to be concise (which is also a philosophy of keyword search processing on relational databases or XML, where a result is defined as a minimal tree that contains all query keywords), we opt to choose the minimal view of each relevant workflow hierarchy that contains at least one match to each query keyword to ensure maximum conciseness.

A *query result* of a keyword search Q on a repository of workflow hierarchies R is defined as the minimal view of each relevant workflow hierarchy that contains matches to all query keywords. In the running example, the view shown in Fig. 1 (b) is the minimal view of the relevant workflow hierarchy in (a).

Query results defined in this way are *informative* and *concise*. A query result is *informative*, as it contains matches to all query keywords as well as the dataflows among them. It is *concise* as it only contains a single *query-driven view* of the smallest size without other unnecessary views of the relevant workflow hierarchy.

B. Query Result Construction

After we have defined the result of keyword search on workflow hierarchies in Section II-A, now we give an overview of how to efficiently generate the results. Note that the minimal view that contains all query keywords generally does not explicitly present in the workflow hierarchy, such as the one in Fig. 1(b), but needs to be dynamically constructed. To construct query results, we categorize the nodes in a relevant workflow hierarchy into three categories with respect to the query keywords, then process them accordingly to generate query results.

1) *Match nodes*: nodes that contain matches to query keywords themselves. Match nodes are explicitly specified by users and therefore are directly output as part of the query result. The properties of match nodes can be displayed upon click.
2) *Expansion nodes*: non-match nodes that have match nodes as their descendants. Since expansion nodes do not directly provide the information specified in a user query, we zoom in them recursively till their descendant match nodes are exposed. [2].
3) *Component nodes*: the rest of the nodes in the workflow hierarchy. Component nodes do not contain keyword themselves; neither do their descendants. Since component nodes are less likely to be interesting to the user, their names are included in the query result only if they are required, together with match nodes, to compose a valid view, but their detailed information is hidden.

Now consider Q on the workflow hierarchy in Fig. 1(a). Since this workflow hierarchy contains matches to all query keywords, it is a relevant workflow hierarchy. Next we show how to find its minimal query-driven view as a query result. We process the nodes in the workflow hierarchy in a top-down fashion. We start with zooming in expansion node `curry chicken` (0). In layer 1, `make chicken broth` (0.0) is a component node whose name should be output in order to compose a valid view, but its descendants will not be visited or output. Since node `preprocess chicken` (0.1) is an expansion node, we zoom in it and recursively process the nodes in the middle box in layer 2. We output match node `tenderize chicken breast` (0.1.0), and then recursively zoom in expansion node `concoct` (0.1.1). When processing node `put into skillet` (0.1.1.0), we notice that there is no explicit dataflow edge between node `tenderize chicken breast` (0.1.0) and `put into skillet` (0.1.1.0). However, based on the ancestor-descendant relationship between node 0.1.1 and 0.1.1.0, this dataflow information can be inferred. Similarly, later on the dataflow from node `saute until tender` (0.1.1.2) to node `stir in flour` (0.2.0) is dynamically derived. Continuing this process, we obtain all the nodes in the minimal view, as shown in Fig. 1(b). The tasks `input` and `output` in the query result are virtual nodes, connecting the dynamically constructed dataflow edges.

III. System Architecture

WISEis implemented using Java, and all the data and indexes are stored in DB2. The architecture of WISEis presented in Fig. III. *Data Parser* and *Index Builder* parse the workflow hierarchies and build indexes, respectively. Upon a user keyword query issued, *Relevant Workflow Hierarchy Finder* retrieves the workflow hierarchies in the repository that contain all query keywords, then *Minimal View Constructor* projects each relevant workflow hierarchy to a two dimensional viewing plane specified by the user query. Finally the *Result Graph Generator* displays the minimal relevant views in a graphical interface.

The *Data Parser* takes input workflow hierarchies in MoML (Modeling Markup Language) [4], the standard format for representing workflows. It parses the input data and assigns labels to nodes and edges in the workflow hierarchy for efficient query processing. Each node in the workflow hierarchy

[2]If two match nodes have an ancestor-descendant relationship, the ancestor match node is treated as an expansion node with an annotation on its descendant nodes.

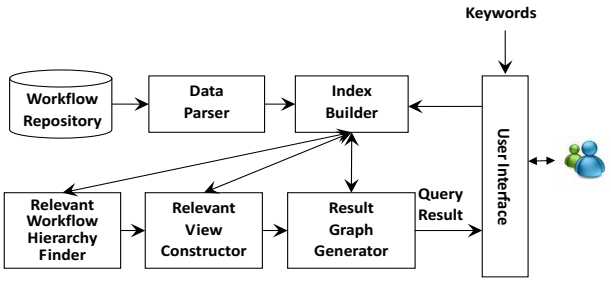


Fig. 2. Architecture of WISE

is assigned a Dewey label as its unique identifer, as shown in Fig. 1(a). Each edge is also assigned with a label such that the dataflows that involve composite tasks can be efficiently identified.

The *Index Builder* constructs an inverted full-text index that maps keywords to nodes in the workflow hierarchy. Another index is built to retrieve the information of a node according to its identifier.

The *Relevant Workflow Hierarchy Finder* retrieves the most specific workflow hierarchy in the repository that contains matches to all the keywords in the user query. It can be efficiently identified by the LCA (Least Common Ancestor) of keyword matches using their Dewey labels. In our running example, node with Dewey lable 0.1.0 matches `chicken breast`, node 0.2.2 matches `coconut milk` and node 0.1.1.2 matches `saute`. Then the most specific workflow hierarchy that contains all matches in Fig. 1(a) is the one named as `curry chicken`, which has a Dewey label 0.

The *Minimal View Constructor* dynamically generates the minimal view of the retrieved relevant workflow hierarchy driven by query specification, as discussed in Section II-B. This process can be performed during a single traversal of the relevant workflow hierarchy.

The *Result Graph Generator* provides a friendly graphical interface to the end users which takes user keywords as input and presents the query results. Users can feel free to drag and drop nodes in the graph.

IV. Demonstration

What will be shown in the demo? We present WISE, a workflow information search engine, which enables effective workflow retrieval and hence facilitates the workflow design and analysis.

WISE has a web-based user interface (http://wise.asu.edu/) which allows users to input keyword searches for retrieving relevant workflows in a repository. The sample workflow repository includes scientific workflows in various domains like biology, chemistry, ecology, provided by Kepler [2], Taverna [6] and myGrid [5], as well as some recipes.

Query results are displayed graphically, similar as shown in Fig. 1(b). Tasks that match keywords are highlighted in the query result, surrounded by a red dashed rectangle. A user can drag and drop tasks on the screen to display the workflow according to his/her preferences. To inform users the dataflows that are dynamically synthesized across multiple layers in the workflow hierarchies, virtual nodes named as `input` and `output` are included in the query result.

Besides the demonstration of the implementation of WISE, we will discuss and justify in more detail about the design choices that we have made on query definition, indexes and query processing algorithms. In addition, performance evaluation on search quality with respect to user studies and on search efficiency over a comprehensive test set will be exhibited.

Significance in database technology. This paper ventures into a quickly expending domain with increasingly substantial data management requirements: workflow management. While the application domain is new, the problem space is not outside of the scope of database research. In particular, database community has extensively investigated techniques for storing, indexing and querying data graphs (e.g. RDBMS) and trees (e.g. XML). This domain, on the other hand, has storing, indexing and querying requirements on a three dimensional data structure consisting of multiple nested graphs. This data structure is a generalization of graphs and trees, which proposes unique technical challenges on data management and call for new solutions.

In this work, we have addressed an open problem of supporting effective keyword search on this type of data structure and tackled several database challenges, including:

- Proposing keyword search result definitions, complementing the keyword search results defined for graph/tree-structured data which are widely adopted in database research.
- Dynamically constructing a query result that is a projection of a relevant workflow hierarchy to the two dimensional viewing plane driven by the user query. Unlike keyword search on graph/tree-structured data which only requires *information retrieval*, we need to perform *information synthesis* to capture dataflows.
- Exploiting labeling schemes and developing indexes to speed up query processing.

The proposed techniques in WISE are general for this type of data structure. We believe studying fundamental database management issues (storage, indexes, labeling schemes, query processing, etc.) on this data structure opens up many opportunities for diverse application domains.

Acknowledgment

We are grateful to K. Selçuk Candan for his constructive and valuable feedback on this work. This research was supported in part by the NSF grant IIS-0740129.

References

[1] S. B. Davidson and J. Freire. Provenance and scientific workflows: challenges and opportunities. In *SIGMOD*, 2008.
[2] Kepler. http://kepler-project.org/.
[3] B. Ludäscher and C. Goble. Guest editors' introduction to the special section on scientific workflows. *SIGMOD Rec.*, 34(3):3–4, 2005.
[4] MoML. http://ptolemy.eecs.berkeley.edu.
[5] myGrid project. http://www.mygrid.org.uk/.
[6] Taverna. http://taverna.sourceforge.net/.