Enabling Ad Hoc Queries over Low-Level Scientific Data Sets

David Chiu and Gagan Agrawal

Department of Computer Science and Engineering
The Ohio State University, Columbus, OH 43210, USA
{chiud,agrawal}@cse.ohio-state.edu

Abstract. Technological success has ushered in massive amounts of data for scientific analysis. To enable effective utilization of these data sets for all classes of users, supporting intuitive data access and manipulation interfaces is crucial. This paper describes an autonomous scientific workflow system that enables high-level, natural language based, queries over low-level data sets. Our technique involves a combination of natural language processing, metadata indexing, and a semantically-aware workflow composition engine which dynamically constructs workflows for answering queries based on service and data availability. A specific contribution of this work is a metadata registration scheme that allows for a unified index of heterogeneous metadata formats and service annotations. Our approach thus avoids a standardized format for storing all data sets or the implementation of a federated, mediator-based, querying framework. We have evaluated our system using a case study from the geospatial domain to show functional results. Our evaluation supports the potential benefits which our approach can offer to scientific workflow systems and other domain-specific, data intensive applications.

1 Introduction

From novel simulations to on-site sensors, advancements in scientific technology have sparked a rapid growth in the deployment of data sources. Vast numbers of low-level data sets, as a result, are persistently stored on distributed disks for access, analysis, and transformation by various classes of users. Managing these low-level data sets on distributed file systems for intuitive user access requires significant consideration towards novel designs in indexing, querying, and integration. At the same time, processes and tools from which the user accesses and manipulates these data sets need to be high-level, if not transparent and automatic. As a result, efforts towards realizing process interoperability and standardized invocation have resulted in the emergence of service-oriented architectures. However, user queries often require a composition of services in able to derive results. And while scientific workflow management systems [1,10,19,22] have made tremendous strides toward scheduling and execution of dependent services, workflows are still largely composed by the user. For scientists and experts, this approach is often sufficient. But in a time when users are becoming

more query- and goal-oriented, solutions which leverage effective use of domain information would require significantly less effort.

In this paper we describe a scientific workflow system which enables high-level queries over low-level data sets. Our system, empowered by a nuanced semantics framework, constructs and executes workflow plans automatically for the user. The approach is motivated by the following observations:

- *The aforementioned trend towards service-oriented solutions in scientific computing.* The data grid community, for instance, has benefitted greatly from borrowing web service standards for interoperability through the Open Grid Service Architecture (OGSA) initiative [13].
- *The trend towards the specification of metadata standards in various scientific domains.* These standards allow for clarity in both user and machine interpretability of otherwise cryptic data sets. However, in many sciences, metadata formats are often heterogeneous, and a unified method to index similar information is lacking, specifically for the purposes of workflow composition.

In our system, data sets are required to be *registered* into an index of metadata information, e.g., time, creator, data quality, etc. A simple domain ontology is superimposed across the available data sets and services for enabling workflow planning. Although traditional database and data integration methods (such as the use of federated databases [24] or mediator-based systems [14,25]) can be applied, our approach does not require a standardized format for storing data sets or the implementation of a complex mediator-based querying framework. Our system combines workflow composition with machine-interpretable metadata, a domain ontology, and a natural language interface to offer simple and intuitive ways for querying a variety of data sets stored in their original low-level formats.

Our experimental evaluation is driven through a case study in the geospatial domain. In computing environments with small numbers of data sets, we show that the benefits of our index-enabled workflow planner is unequivocally apparent. Moreover, the scalability of these benefits are easily observable for larger numbers of indices and data sets. Overall, we show that workflows can be composed efficiently using data sets described in disparate metadata formats.

The remainder of this paper is organized as follows: An overview of our system is presented in Section 2. Specifications of majors components in our system are discussed in Section 3. Section 3.1 describes the query decomposition process, followed by metadata registration in Section 3.2. The workflow planning algorithm is presented in Section 3.3. Section 4 explains the results of our performance evaluation on a case study in the geospatial domain. We compare our work with related efforts in Section 5, and finally conclude in Section 6.

2   System Overview

The system architecture, shown in Figure 1, consists of four independent layers. The design and implementation of each layer can be superseded without affecting
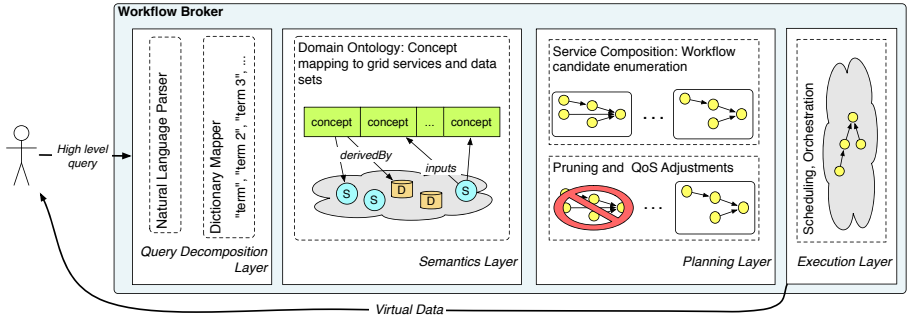
Fig. 1. System Architecture

the others as long as it subscribes to some system-specified protocols. From a high-level perspective, our system can be viewed as a *workflow broker:* as users submit queries to the system, the broker plans and executes the workflows involved in deriving the desired virtual data while hiding complexities such as workflow composition and domain knowledge from the user.

The user specifies queries through the broker's *Query Decomposition Layer.* This layer decomposes a natural language-based query into keywords and grammatical dependencies using a natural language parser. The parsed set of keywords is then mapped to concepts within the domain ontology specified in the next layer. The *Semantics Layer* maintains an active list of available services, data sets, and their metadata. While the Web Service Description Language (WSDL) [6] is the international standard for web service description, scientific data sets often lack a singular metadata format. For instance, in the geospatial domain alone, CSDGM (Content Standard for Digital Geospatial Metadata) [12] is adopted in the United States. Elsewhere, Europe and Australia have proposed similar metadata standards. More important, XML essentially opens the possibility for any user to define any descriptive data annotation, at any time. But while their formats differ in specification, the information captured is similar: dates of relevance, spatial region, data quality, etc. In the next section, we discuss ways our system deals with heterogeneous metadata.

Although metadata is imperative for providing descriptions for data sets and services, a higher level descriptor is also needed to define the relationships between the available data and services to concepts within some scientific domain. These relationships help facilitate planning algorithms for workflow composition. For example, there is a need for the system to understand how "water levels" are derived using some existing data sets, services, or combinations of both. We specify this description through a simple ontology, a directed graph with the following requirements:

– The ontology consists of three disjoint sets (classes) C, S, and D representing the set of domain concepts, the set of available services known to the system, and the known domain-specific data types, respectively.
– Two types of directed edges (relations) exist: concepts may be *derivedFrom* data or service nodes and a service *inputsFrom* concepts.
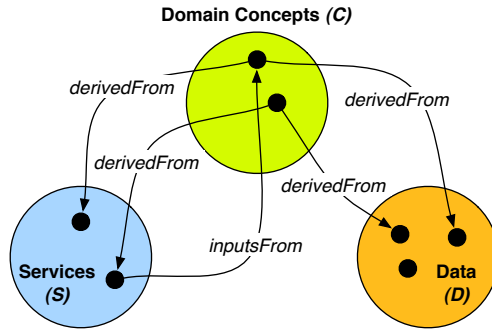
Fig. 2. Ontological Structure for Domain Description

This ontological definition, shown in Figure 2, simplifies the effort to indicate which services and data types are responsible for deriving specific domain concepts.

Next, the *Planning Layer* assumes that the ontology and metadata index are in place and defined. The planning algorithm, discussed in detail in the following section, relies heavily on the Semantics Layer. In essence, the planner enumerates workflows to answer any particular query through traversals of the domain ontology. The existence of needed services and data sets is identified by the metadata index. This layer sends a set of workflows all capable of answering the user query to the *Execution Layer* for processing, and the resulting virtual data is finally returned back to the user.

In this paper we focus mainly on the Query Decomposition and Semantics Layers. While the workflow enumeration algorithm in the Planning Layer is also described, details on the algorithm's cost-based pruning mechanism, QoS adaptation, and robustness over distributed environments are discussed in our other publications [8,7].

3   Technical Details

This section focuses on the descriptions of the major components involved in supporting queries, semantics, and planning. We lead into the discussion of the technical specifications of each system component through a simple working example query.

```
''return water level from station=32125 on 10/31/2008''
```

3.1   Query Decomposition

The first objective of our system is to process user queries in the form of natural language. The job of the Query Decomposition Layer is to extract relevant elements from the user query. These elements, including the user's desiderata
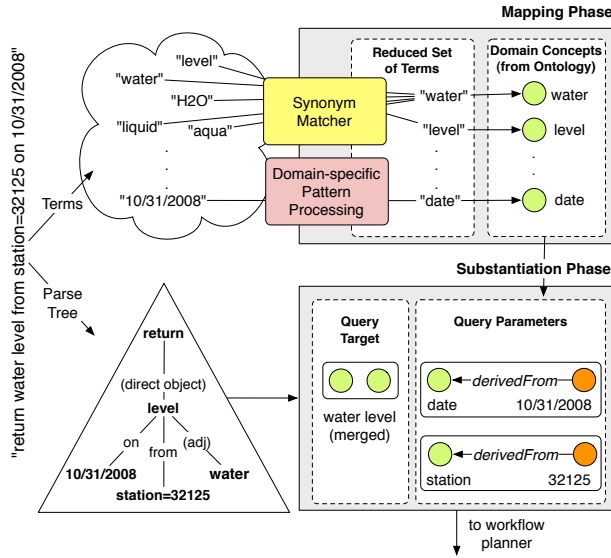
Fig. 3. Query Decomposition Process

and other query attributes, are mapped to domain concepts specified in the Semantics Layer's ontology. Thus, these two layers in the system architecture are tightly linked. Shown in Figure 3, the decomposition process is two-phased.

In the Mapping Phase, StanfordNLP [17] is initially employed to output a list of terms and a parse tree from the query. The list of extracted query terms is then filtered through a stop list to remove insignificant terms. This filtered set is further reduced using a synonym matcher provided through WordNet libraries [11]. The resulting term set is finally mapped to individual domain concepts from the ontology. These terms, however, can also take on meaning by their patterns. In our example, "10/31/2008" should be mapped to the domain concept, *date*. A pattern-to-concept matcher, for this reason, was also implemented using regular expressions. But since only certain patterns can be anticipated, some querying guidelines must be set. For instance, dates must be specified in the *mm/dd/yyyy* format, time as *hh:mm:ss*, coordinates as *(lat, long)*, etc. Additionally, values can also be given directly to concepts using a *concept=value* string, as seen for assigning 32125 to *station* in our query.

Upon receiving the set of relevant concepts from the previous phase, the Substantiation Phase involves identifying the user's desired concept as well as assigning the given values to concepts. First, from the given parse tree, concepts are merged with their descriptors. In our example, since "water" describes the term "level", their respective domain concepts are merged. The pattern matcher from the previous phase can be reused to substantiate given values to concepts, resulting in the relations *(date derivedFrom 10/31/2008)* and *(station derivedFrom 32125)*. These query parameter substantiations is stored as a hashset, $Q[\ldots] = Q[date] \rightarrow \{10/31/2008\}$ and $Q[station] \rightarrow \{32125\}$. This set of query

parameters is essential for identifying accurate data sets in the workflow planning phase. Query parameters and the target concept, are sent as input to the workflow planning algorithm in the Planning Layer of the system.

We take prudence in discussing our query parser as not to overstate its functionalities. Our parser undoubtedly lacks a wealth of established natural language query processing features (see Section 5), for it was implemented ad hoc for interfacing with our specific domain ontology. We argue that, while related research in this area can certainly be leveraged, the parser itself is ancillary to meeting the system's overall goals of automatic workflow planning and beyond the current scope of this work. Nonetheless, incorrectly parsed queries should be dealt with. Currently, with the benefit of the ontology, the system can deduce the immediate data that users must provide as long as the target concept is determined. The user can then enter the required data into a form for querying.

3.2   Metadata Registration

Because workflow planning is a necessary overhead, the existence of data sets (and services) must be identified quickly. Our goal, then, is to provide fast data identification. On one hand, we have the challenge of supplying useful domain knowledge to the workflow planner, and on the other, we have a plethora of pre-existing database/metadata management technologies that can be leveraged. The result is to utilize an underlying database to store and index domain-specific elements and, with the advantage of fast indices, the overhead of data identification for workflow planning can be optimized. For each data set, its indexed domain concepts can be drawn from an accompanying metadata file. However, metadata formats for describing scientific data sets can vary. There exists, for instance, multiple annotation formats from just within the geospatial community. But while their structures differ, the descriptors are similar, storing essential information (data quality, dates, spatial coverage, and so on) pertaining to specific data sets.

Domain experts initialize the system with the following[1]: (i) $\Upsilon = \{v_1, \ldots, v_n\}$, a set of XML Schema or Data Type Definitions (DTD) which defines the supported metadata formats used for validation. (ii) C_{idx}, a set of domain concepts that the system should index, and (iii) $xpath(v, c) : (v \in \Upsilon \wedge c \in C_{idx})$, For each indexed concept and schema, an XPath query [5] that is used to access the indexed value for concept c from a given the metadata document corresponding to schema v. Once in place, domain users should be able to upload and not only share new data sets, but to also make it available for answering high-level queries. To *register* new data sets with the system, users can invoke the Data Registration algorithm. This procedure, shown in Algorithm 1, takes three inputs: a data file d, its metadata file $meta_d$, and an optional keyword array, $K[\ldots]$ that describes d, and an optional schema for validating $meta_d$, v_d. With the benefit of the term-to-concept mapper described in Section 3.1, the domain concept to which this data set derives can be computed. Optionally, but not shown, the

---

[1] Our implementation assumes that metadata is defined in XML.

Algorithm 1. registerData(d, $meta_d[$, K, $v_d]$)

1: /* identify and validate metadata */
2: **if** $v_d \in \Upsilon \wedge v_d$.validate($meta_d$) = true **then**
3:     $\delta \leftarrow v_d$ /* input schema checks out */
4: **else**
5:     **for all** $v \in \Upsilon$ **do**
6:         **if** v.validate($meta_d$) = true **then**
7:             $\delta \leftarrow v$ /* δ holds the corresponding schema */
8:         **end if**
9:     **end for**
10: **end if**
11: $c_K \leftarrow ConceptMapper$.map($K$) /* solve for concept derived by d */
12: $d_K \leftarrow c_K \cdot$ "type"
13: **if** $\nexists\, d_K \in Ontology.D$ **then**
14:     $Ontology.D \leftarrow Ontology.D \cup \{d_K\}$
15:     $Ontology.Edges \leftarrow Ontology.Edges \cup \{(c_K, derivedFrom, d_K)\}$
16: **end if**
17: /* build database record */
18: $R \leftarrow (datatype = d_K)$
19: **for all** $c \in C_{idx}$ **do**
20:     $v \leftarrow meta_d$.extract($xpath(\delta, c)$)
21:     $R \leftarrow record \cup (c = v)$ /* concatenate record */
22: **end for**
23: DB.insert(R, d)

---

user could select concepts directly from the ontology to describe d's type instead of providing $K[...]$. To avoid confusion of schema validity and versioning, we emphasize here that the set of valid schemas, Υ, should only be managed by domain experts or administrators. That is, although users may potentially discover new metadata schemas, our system cannot allow them to update Υ directly.

Algorithm 1 starts by identifying the type of metadata, δ, prescribed by the user via validating $meta_d$ against the set of schemas, or directly against the user provided schema, v_d (Lines 2-10). Next, the domain concept that is represented by $K[...]$ is solved for on Line 11. On Line 12, d_K is assigned the name representing the type of data in the ontology, where c_K is the matched concept and $\cdot$ denotes string concatenation. If necessary, d_K is added into the ontology's data type class, D, and an edge from c_K to d_K is also established (Lines 13-16). Finally, on Lines 18-23, a record is constructed for eventual insertion into the underlying database. The constructed record, R, is inserted into the database with a pointer to the data set, d. This process is illustrated in Figure 4(a).

Service registration, depicted in Figure 4(b), is not unlike data registration. First, the service description file (in WSDL [6]) is input. Each declared operation must input domain concepts describing its parameters and output. Again, the user may input keywords defining the concept or map the concepts directly. From concept mapping, the ontology is updated with the respective *derivedFrom* and *inputsFrom* edges. Additionally, preconditions and prediction models for
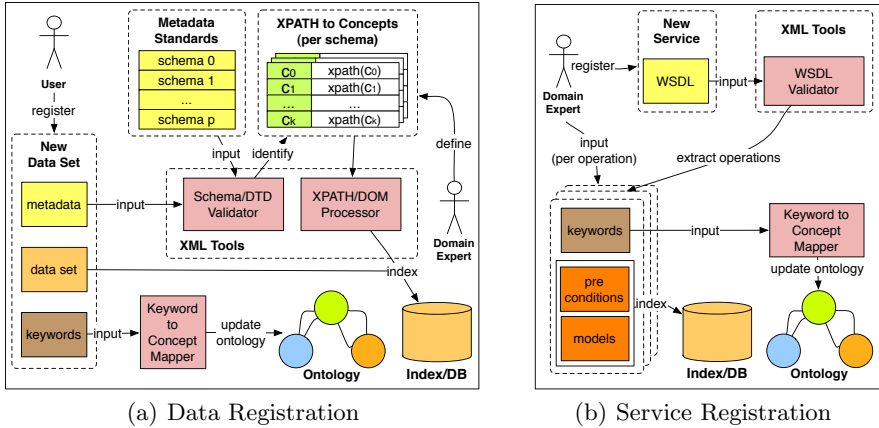
(a) Data Registration   (b) Service Registration

Fig. 4. Metadata Registration Process

execution time and output size can also be input in this stage. Preconditions, useful for workflow planning, and prediction models, required for QoS adaptation, are both indexed per service operation.

3.3 Workflow Planning

Often in practice, scientific tasks are composed of disparate processes chained together to produce some desired values. Although workflows are rooted in business processes, their structures lend well to the realization of complex scientific computing [10,22,1,19]. Workflows can be expressed as directed acyclic graphs where the vertices denote processes/services and data sets and directed edges represent the flow of data. In our framework, we define workflow as follows. Given some arbitrary dataset D and a set of services S a workflow:

$$w = \begin{cases} \epsilon \\ d \\ (s, P_s) \end{cases}$$

such that terminals ϵ and $d \in D$ denote a null workflow and a data instance respectively. Nonterminal $(s, P_s) \in S$ is a tuple where s denotes a service operation with an ordered parameter list $P_s = (p_1, \ldots, p_k)$ and each p_i is itself a workflow. In other words, a workflow is a tuple which either contains a single data instance or a service whose parameters are, recursively, (sub)workflows.

Workflow Enumeration Algorithm. Given some query q, the goal of workflow planning algorithm is to enumerate a list of workflows $W_q = (w_1, \ldots, w_n)$ capable of answering q from the available services and data sets. The execution of each $w_i \in W_q$ is carried out, if needed, by an order determined by cost or

QoS parameters. Thus, upon workflow execution failure, the system can persistently attempt alternative, albeit potentially less optimal (with respect to QoS parameters), workflows. Mechanisms for assigning cost to workflows against QoS constraints, however, are out of the scope for this paper.

Domain concept derivation is the goal behind constructing each workflow. Thus, our algorithm, WFEnum, relies heavily on the metadata and semantics provided in the *Semantics Layer*. Recall from Section 3.1 that the Query Decomposition component outputs the query's target concept, t, and a hashed set of query parameters, $Q[\ldots]$ (such that $Q[concept] \rightarrow \{val_1, val_2, \ldots\}$). The WFEnum algorithm takes both t and $Q[\ldots]$ as input, and outputs a list W of distinct workflows that are capable of returning the desiderata for the target concept.

WFEnum, shown in Algorithm 2, begins by retrieving all $d \in D$ (types of data registered in the ontology) from which the target concept, t, can be derived. On Line 2, a statically accessible array, $W'[\ldots]$, is used for storing overlapping workflows to save redundant recursive calls in the later half of the algorithm. The workflows are memoized on a hash value of their target concept and parameter list. On Line 5, a set of indexed concepts, C_{idx}, is identified for each data type, and checked against the parsed user specified values in the query. To perform this check, if the set difference between the registered concepts, C_{idx}, and the query parameters, $Q[\ldots]$, is nonempty, then the user clearly did not provide enough information to plan the workflow unambiguously. On Lines 7-11, if all index registered concepts are substantiated by elements within $Q[\ldots]$, a database query is designed to retrieve the relevant data sets. For each indexed concept c, its *(concept=value)* pair, $(c = Q[c])$ is concatenated (AND'd) to the query's conditional clause. On Lines 12-15, the constructed query is executed and each returned file record, f, is an independent file-based workflow deriving t.

The latter half of the algorithm deals with concept derivation via service calls. From the ontology, a set of relevant service operations, Λ_{srvc} is retrieved for deriving t. For each operation, *op*, there may exist multiple ways to plan for its execution because each of its parameters, p , is a subproblem. Therefore, workflows pertaining to each parameter p must first be solved with its own target concept, *p.target* and own subset of relevant query parameters $Q_p[\ldots]$. While *p.target* is easy to identify from following the *inputsFrom* links belonging to *op* in the ontology, the forwarding of $Q_p[\ldots]$ requires a bit more effort. Looking past Lines 25-31 for now, this query parameter forwarding process is discussed in detail in Section 3.3.

Once the $Q_p[\ldots]$ is forwarded appropriately, the recursive call can be made for each parameter, or, if the call is superfluous, the set of workflows can be retrieved directly (Line 32-36). In either case the results are stored in W_p, and the combination of these parameter workflows in W_p is established through a cartesian product of its derived parameters (Line 37). For instance, consider a service workflow with two parameters of concepts a and b: $(op, (a, b))$. Assume that target concepts a is derived using workflows $W_a = (w_1^a, w_2^a)$ and b can only be derived with a single workflow $W_b = (w_1^b)$. The distinct parameter

Algorithm 2. WFEnum(t, $Q[\ldots]$)

1: $W \leftarrow ()$
2: global $W'[\ldots]$ /* static table for memoization */
3: $\Lambda_{data} \leftarrow Ontology.\text{derivedFrom}(D, t)$
4: **for all** $d \in \Lambda_{data}$ **do**
5:     $C_{idx} \leftarrow d.\text{getIndexConcepts}()$
6:     /* user-given values enough to substantiate indexed concepts */
7:     **if** $(Q.\text{concepts}() - C_{idx}) = \{\}$ **then**
8:         $cond \leftarrow (datatype = d)$
9:         **for all** $c \in C_{idx}$ **do**
10:             $cond \leftarrow cond \wedge (c = Q[c])$ /* concatenate new condition */
11:         **end for**
12:         $F \leftarrow \sigma_{<cond>}(datasets)$ /* select files satisfying $cond$ */
13:         **for all** $f \in F$ **do**
14:             $W \leftarrow (W, (f))$
15:         **end for**
16:     **end if**
17: **end for**
18:
19: $\Lambda_{srvc} \leftarrow Ontology.\text{derivedFrom}(S, t)$
20: **for all** $op \in \Lambda_{srvc}$ **do**
21:     $\Pi_{op} \leftarrow op.\text{getPreconditions}();$
22:     $P_{op} \leftarrow op.\text{getParameters}()$
23:     $W_{op} \leftarrow ()$
24:     **for all** $p \in P_{op}$ **do**
25:         /* forward query parameters s.t. preconditions are not violated */
26:         $Q_p[\ldots] \leftarrow Q[\ldots]$
27:         **for all** $(concept, value) \in Q_p[\ldots]$ **do**
28:           **if** $(concept, value).\text{violates}(\Pi_{op})$ **then**
29:              $Q_p[\ldots] \leftarrow Q_p[\ldots] - (concept, value)$
30:           **end if**
31:         **end for**
32:         **if** $\exists W'[h(p.target, Q_p)]$ **then**
33:           $W_p \leftarrow W'[h(p.target, Q_p)]$ /* recursive call is redundant */
34:         **else**
35:           $W_p \leftarrow$ WFEnum($p.target$, $Q_p[\ldots]$) /* recursively invoke for p */
36:         **end if**
37:         $W_{op} \leftarrow W_{op} \times W_p$ /* cartesian product */
38:     **end for**
39:     /* couple parameter list with service operation and concatenate to W */
40:     **for all** $pm \in W_{op}$ **do**
41:         $W \leftarrow (W, (op, pm))$
42:     **end for**
43: **end for**
44: $W'[h(t, Q_p)] \leftarrow W$ /* memoize */
45: **return** W

list plans are thus obtained as $W_{op} = W_a \times W_b = ((w_1^a, w_1^b), (w_2^a, w_1^b))$. Each element from W_{op} is a unique parameter list. These lists are coupled with the service operation, op, memoized in W' for avoiding redundant recursive calls in the future, and returned in W (Lines 39-45). In our example, the final list of workflows is obtained as $W = ((op, (w_1^a, w_1^b)), (op, (w_2^a, w_1^b)))$.

The returned list, W, contain planned workflows capable of answering an original query. Ideally, W should be a queue with the "best" workflows given priority. Mechanisms identifying the "best" workflows to execute, however, depends on the user's preferences. Our previous effort have led to QoS-based cost scoring techniques leveraging on bi-criteria optimization: workflow execution time and result accuracy. Although not shown in this paper, the gist of this effort is to train execution time models and also allow domain experts to input error propagation models per service operation. Our planner, when constructing workflows, invoke the prediction models based on user criteria. Workflows not meeting either constraint are pruned on the a priori principle during the enumeration phase. In the special case of when W is empty, however, a re-examination of pruned workflows is conducted to dynamically adapt to meet these constraints through data reduction techniques. This QoS adaptation scheme is detailed in other publications [8,7].

Forwarding of Query Parameters. It was previously noted that planning a service operation is dependent on the initially planning of the operation's parameters. This means that WFEnum must be recursively invoked to plan (sub)workflows for each parameter. Whereas the (sub)target concept is clear to the system from *inputsFrom* relations specified in the ontology, the original query parameters must be forwarded correctly. For instance, consider some service-based workflow, $(op, (L_1, L_2))$ that expects as input two time-sensitive data files: L_1 and L_2. Let's then consider that op makes the following two assumptions: (i) L_1 is obtained at an earlier time/date than L_2 and (ii) L_1 and L_2 both represent the same spatial region. Now assume that the user query provides two dates, $10/2/2007$ and $12/3/2004$ and a location (x, y), that is,

$$Q[\ldots] = \begin{cases} location \to \{(x, y)\} \\ date \to \{10/2/2007, 12/3/2004\} \end{cases}$$

To facilitate this distribution, the system allows a set of preconditions, Π_{op}, to be specified per service operation. All conditions from within Π_{op} must be met before allowing the planning/execution of op to be valid, or the plan being constructed is otherwise abandoned. In our case, the following preconditions are necessary to capture the above constraints:

$$\Pi_{op} = \begin{cases} L_1.date \preceq L_2.date \\ L_1.location = L_2.location \end{cases}$$

In Lines 25-31, our algorithm forwards the values accordingly down their respective parameter paths guided by the preconditions, and thus implicitly satisfying

them. The query parameter sets thus should be distributed differently for the recursively planning of L_1 and L_2 as follows:

$$Q_{L_1}[\ldots] = \begin{cases} location \rightarrow \{(x,y)\} \\ date \rightarrow \{12/3/2004\} \end{cases} \quad Q_{L_2}[\ldots] = \begin{cases} location \rightarrow \{(x,y)\} \\ date \rightarrow \{10/2/2007\} \end{cases}$$

The recursive planning for each (sub)workflow is respectively supplied with the reduced set of query parameters to identify only those files adhering to preconditions.

4 System Evaluation

The experiments that we conducted are geared towards exposing two particular aspects of our system: (i) we run a case study from the geospatial domain to display its functionality, including metadata registration, query decomposition, and workflow planning. (ii) We show scalability and performance results of our query enumeration algorithm, particularly focusing on data identification.

Experimental Case Study. To present our system from a functional stand-point, we employ an oft-used workflow example from the geospatial domain: shoreline extraction. This application requires a Coastal Terrain Model (CTM) file and water level information at the targeted area and time. CTMs are essentially matrices (from a topographic perspective) where each point represents a discretized land elevation or bathymetry (underwater depth) value in the captured coastal region. To derive the shoreline, and intersection between the effective CTM and a respective water level is computed. Since both CTM and water level data sets are spatiotemporal, our system must not only identify the data sets efficiently, but plan service calls and their dependencies accurately and automatically.

For this example, the system's data index is configured to include only *date* and *location* concepts. In practice however, it would be useful to index additional elements such as resolution/quality, creator, map projection, and others. Next, we provided the system with two metadata schemas, the U.S.-based CSDGM [12] and the Australia and New Zealand standard, ANZMETA [3], which are both publicly available. Finally, XPaths formed from the schemas to index concepts *date* and *location* for both schemas are defined.

Next, CTM files, each coupled with corresponding metadata and keywords $K = \{$ *"CTM", "coastal terrain model", "coastal model"*$\}$, are inserted into the system's registry using the data registration procedure provided in Algorithm 1. In the indexing phase, since we are only interested in the spatiotemporal aspects of the data sets, a single modified B^x-Tree [16] is employed as the underlying database index for capturing both *date* and *location*.[2] For the ontology phase,

---

[2] Jensen et al.'s B^x-Tree [16], originally designed for moving objects, is a B+Tree whose keys are the approximate linearizations of time and space of the object via space-filling curves.
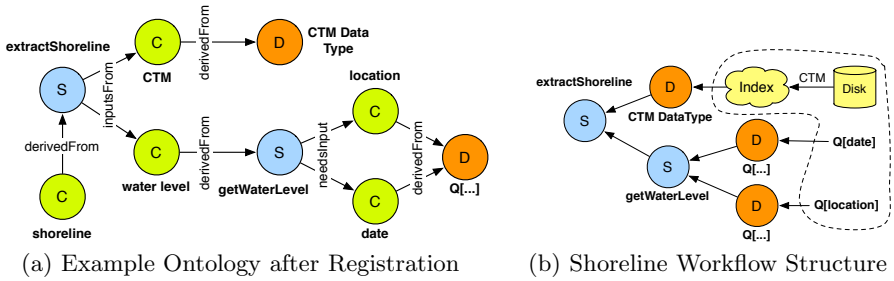
(a) Example Ontology after Registration   (b) Shoreline Workflow Structure

Fig. 5. The Shoreline's Involved Ontology and the Derived Workflow

since a *CTM* concept is not yet captured in the domain ontology, the keyword-to-concept mapper will ask the user to either (a) display a list of concepts, or (b) create a new domain concept mapped from keywords K. If option (a) is taken, then the user chooses the relevant concept and the incoming data set is registered into the ontology, and K is included the mapper's dictionary for future matches. Subsequent CTM file registrations, when given keywords from K, will register automatically under the concept *CTM*. On the service side, two operations are required for registration, shown below as $(op, (c_{p1}, c_{p2}, \ldots, c_{pk}))$, where *op* denotes the service operation name and c_{pi} denotes the domain concept of parameter i:

1. *(getWaterLevel, (date, location))*: retrieves the average water level reading on the given date from a coastal gauging station closest to the given location.
2. *(extractShoreline, (CTM, water level))*: intersects the given CTM with the water level and computes the shoreline.

For sake of simplicity, neither operation requires preconditions and cost prediction models. After metadata registration, the resulting ontology is shown in Figure 5(a), unrolled for clarity. Albeit that there are a multitude of more nodes in a practical system, it is easy to see how the WFEnum algorithm would plan for shoreline workflows. By traversing from the targeted concept, *shoreline*, and visiting all reachable nodes, the workflow structure is a reduction of *shoreline*'s reachability subgraph with a reversal of the edges and a removal of intermediate concept nodes. The abstract workflow shown in Figure 5(b) is the general structure of all plannable workflows. In this particular example, WFEnum will enumerate more than one workflow candidate only if multiple CTM files (perhaps of disparate resolutions) are registered in the index at the queried location and time.

Performance Evaluation. Our system is distributed by nature, and therefore, our testbed is structured as follows. The workflow planner, including metadata indices and the query parser, is deployed onto a Linux machine running a Pentium 4 3.00Ghz Dual Core with 1GB of RAM. The geospatial processes are deployed as web services on a separate server located across the Ohio State University campus at the Department of Civil and Environmental Engineering and Geodetic Science.
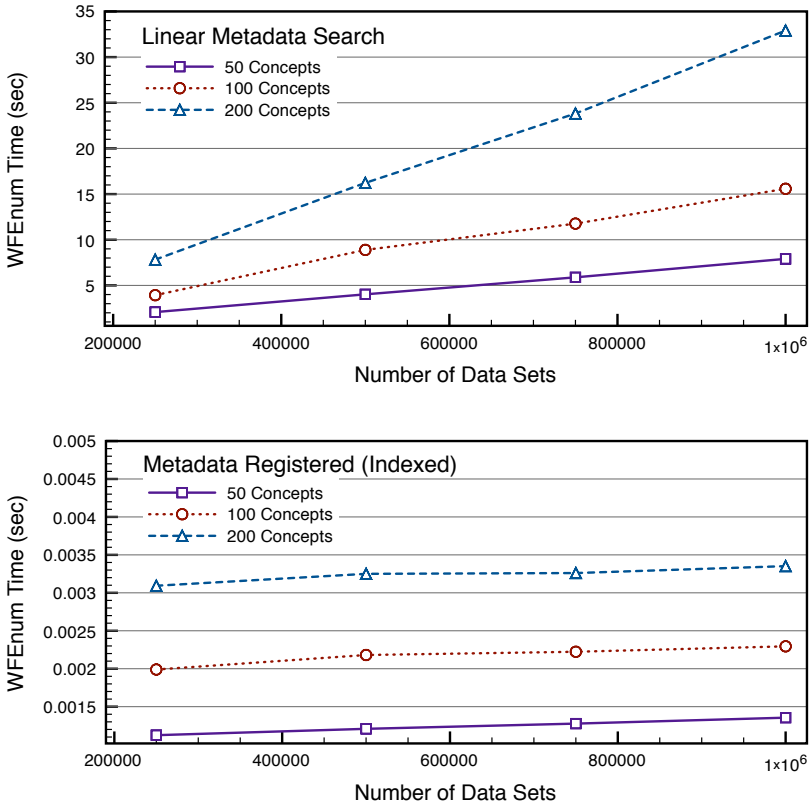
Fig. 6. Workflow Planning Times with Increasing Data Sets and Concept Indices

CTM data sets, while indexed on the workflow planner node, are actually housed on a file server across state, at the Kent State University campus.

In the first experiment, we are interested in the runtime of WFEnum with and without the benefit of metadata registration when scaled to increasing amounts of data files and concepts needing indexed (thus resulting in both larger index structures and a larger number of indices). Shown in Figure 6 (top), the linear search version consumes significant amounts of time, whereas its counterpart (bottom) consumes mere milliseconds for composing the same workflow plan. Also, because dealing with multiple concept indices is a linear function, its integration into linear search produces drastic slowdowns. And although the slowdown can also be observed for the indexed runtime, they are of negligible amounts.

Once the shoreline extraction workflow has finished planning, its execution is then carried out by our system. As seen in Figure 7, the workflow's execution time is heavily dependent on the CTM file size. If we juxtaposed Figure 6 with Figure 7, the importance of minimizing planning time becomes clear. Especially for smaller CTM files, the cases when planning times dominate execution
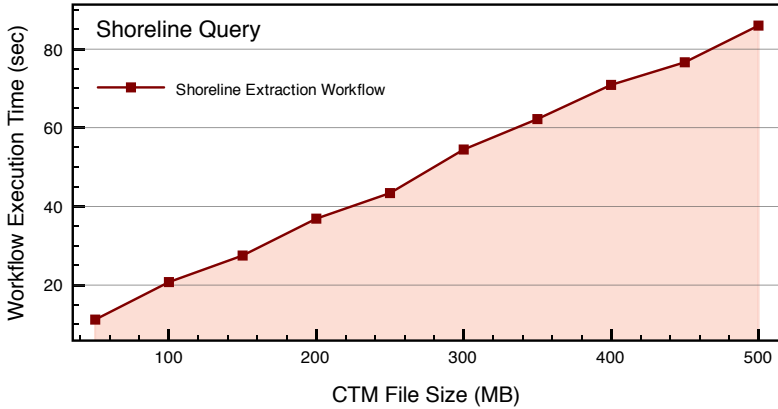
Fig. 7. Shoreline Workflow Execution Times

times should be avoided, and metadata indexing decreases the likelihood for this potential.

5   Related Efforts

The need for metadata and semantics has long been addressed by such initiatives as the plethora of XML-based technologies, including Resource Description Framework (RDF) and its complement, the Web Ontology Language (OWL) [20,9]. These standards have opened up support to allow anyone to specify human and machine interpretable descriptions for any type of data. In our system, we indeed employ RDF+OWL to formalize a general ontology which describes the relationships between concepts and resources (data sets and services). This resource description is imperative to our system, as it semantically drives the workflow planner.

Parsing correctness, disambiguation, and schema mapping are well-known problems in natural language querying. Stanford's Natural Language Parser [17] and dictionary API's provided by WordNet [11] are often employed in systems providing natural language support, including our own. In the direction of querying structured information, ample research has been developed for addressing the issues with translating natural language translation to structured queries [2,18]. We concede that our parser is lacking such relevant technologies for handling the age-old challenges of disambiguation, mapping, etc. Undertaking the implementation of these features is currently beyond the scope of this work.

Research in high performance scientific data management has produced such systems as the Scientific Data Manager (SDM), which employs the Meta-data Management System (MDMS) [21]. SDM provides a programming model and abstracts low-level parallel I/O operations for complex scientific processing. While MDMS uses a database for metadata storage, the metadata itself is specific to the scientific process at hand, containing information on execution (e.g., access

patterns, problem size, data types, file offsets, etc). This metadata is used by SDM to optimize the runtime of these parallel processes. Another system, San Diego Supercomputing Center's Storage Resource Broker (SRB) [4], seeks to store massive volumes of data sets split across clusters or nodes within heterogenous environments. SRB allows parallel and transparent data access by offering a simplified API to users which hides complexities such as merging data sets, allowing restricted access, etc. Compared to our system, there is a fundamental difference in functionality. Ours provides a way to store heterogeneous metadata specific to scientific domains inside a database, and that the metadata are invoked not for process optimization, but for data identification purposes for automatic workflow planning.

Ways to handle the heterogeneity of metadata have prompted many works on metadata cataloguing and management. Particularly, in the volatile grid computing environment, data sources are abundant and metadata sources are ever-changing. Metadata Catalog Service (MCS) [26] and Artemis [28] are collaborative components used to access and query repositories based on metadata attributes. MCS is a self-sufficient catalog which stores information on data sets. Its counterpart Artemis, on the other hand, can be used to integrate many versions of MCS for answering interactive queries. Their interface takes users through a list of questions guided by a domain ontology to formulate a query. The planned query is then sent to the Artemis mediator to search for relevant items in the MCS instances. While the MCS and Artemis is somewhat tantamount to our metadata registration and automatic query formulation processes, our systems differ in the following ways. (i) Ours not only facilitates accurate data identification based on metadata querying, but also combining these data items with similarly registered services to compose workflows. (ii) Although both systems allow higher level querying frameworks, our approach is enabled through natural language and keyword mapping of domain ontology concepts.

Workflow management systems have also gained momentum in the wake of managing complex, user-driven, scientific computations in the form of service composition. By itself, service composition have become prevalent enough to warrant such industrial standards as the WSBPEL (Web Service Business Process Execution Language) [30] to describe the orchestration of service execution. Implementations of WSBPEL engines have already sprawled into realms of proprietary and open-source communities, an auspicious indication of the high optimism for the movement toward service-oriented workflow solutions. In fact, many efforts towards scientific workflow composition have already been developed. Notable systems such as Askalon [29], Taverna [22], and Kepler [1] have evolved into grid- and service-oriented systems. These systems typically allow domain experts to define static workflows through a user-friendly interface, and map the component processes to known services. Pegasus [10,15] allows users to compose workflows potentially with thousands of nodes using abstract workflow templates. But while these systems alleviate users' efforts for composition, our system proposes automatic workflow planning based on available metadata to elude user-based composition altogether.

In the direction of automatic workflow composition, Traverso et al. discussed the importance of exploiting semantic and ontological information for automating service composition [27]. Their approach generates automata-based "plans," which can then be translated into WSBPEL processes. The goals and requirements for these plans, however, must be expressed in a formal language, which may be cryptic for the average user. Other automatic planning-based systems, such as Sword [23] and SHOP2 [31], also require similar complexity in expressing workflows. While the overall objectives of these systems are tantamount to those of our own, our directions are quite different. In an age when scientific data sets are ubiquitous and when machine- and human-interpretable descriptions are imperative, we are invariably deluged with high-volumes of heterogeneous data sets and metadata. To the best of our knowledge, the registration and exploitation of domain-specific metadata to automatically compose workflows for answering natural language queries is a novel approach in this area.

6   Conclusion and Future Work

In this paper we have presented a system which supports simplified querying over low-level scientific datasets. This process is enabled through a combination of effective indexing over metadata information, a system and domain specific ontology, and a workflow planning algorithm capable of alleviating all tiers of users of the difficulties one may experience through dealing with the complexities of scientific data. Our system presents a new direction for users, from novice to expert, to share data sets and services. The metadata, which comes coupled with scientific data sets, is indexed by our system and exploited to automatically compose workflows in answering high level queries without the need for common users to understand complex domain semantics.

As evidenced by our experiments, a case can be made for supporting metadata registration and indexing in an automatic workflow management system. In our case study alone, comparing the overhead of workflow planning between linear search and index-based data identification methods, speedups are easily observed even for small numbers of data sets. Further, on the medium scale of searching through 1×10^6 data sets, it clearly becomes counterproductive to rely on linear metadata search methods, as it potentially takes longer to plan workflows than to execute them. As evidenced, this scalability issue is easily mitigated with an indexed approach, whose planning time remains negligible for the evaluated sizes of data sets.

Although our system claims to support natural language queries, it is, admittedly, far from complete. For instance, mapping sophisticated query elements to supporting range queries and joins is lacking. While this limits querying support significantly, we believe that these details can be realized with more effort spent on providing better models in the relationship between parse trees and the query plan. Nonetheless, in a more holistic sense, these nuances are diminutive against the general role of the system.

Acknowledgments

References

1. Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludscher, B., Mock, S.: Kepler: An extensible system for design and execution of scientific workflows (2004)
2. Androutsopoulos, I., Ritchie, G.D., Thanisch, P.: Natural language interfaces to databases–an introduction. Journal of Language Engineering 1(1), 29–81 (1995)
3. ANZLIC. Anzmeta xml document type definition (dtd) for geospatial metadata in australasia (2001)
4. Baru, C., Moore, R., Rajasekar, A., Wan, M.: The sdsc storage resource broker. In: CASCON 1998: Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research, p. 5. IBM Press (1998)
5. Berglund, A., Boag, S., Chamberlin, D., Fernndez, M.F., Kay, M., Robie, J., Simon, J.: Xml path language (xpath) 2.0. w3c recommendation (January 23, 2007), http://www.w3.org/tr/xpath20
6. Chinnici, R., Moreau, J.-J., Ryman, A., Weerawarana, S.: Web services description language (wsdl) 2.0. w3c recommendation (June 26, 2007), http://www.w3.org/tr/wsdl20/
7. Chiu, D., Deshpande, S., Agrawal, G., Li, R.: Composing geoinformatics workflows with user preferences. In: GIS 2008: Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems. ACM, New York (2008)
8. Chiu, D., Deshpande, S., Agrawal, G., Li, R.: Cost and accuracy sensitive dynamic workflow composition over grid environments. In: Proceedings of the 9th IEEE/ACM International Conference on Grid Computing (Grid 2008) (2008)
9. Dean, M., Schreiber, G.: Owl web ontology language reference. w3c recommendation (2004)
10. Deelman, E., Singh, G., Su, M.-H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G.B., Good, J., Laity, A.C., Jacob, J.C., Katz, D.S.: Pegasus: A framework for mapping complex scientific workflows onto distributed systems. Scientific Programming 13(3), 219–237 (2005)
11. Fellbaum, C. (ed.): WordNet: An Electronic Lexical Database. MIT Press, Cambridge (1998)
12. FGDC. Metadata ad hoc working group. content standard for digital geospatial metadata (1998)
13. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The physiology of the grid: An open grid services architecture for distributed systems integration (2002)
14. Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J., Widom, J.: Integrating and Accessing Heterogenous Information Sources in TSIMMIS. In: Proceedings of the AAAI Symposium on Information Gathering (1995)
15. Gil, Y., Ratnakar, V., Deelman, E., Mehta, G., Kim, J.: Wings for pegasus: Creating large-scale scientific applications using semantic representations of computational workflows. In: Proceedings of the 19th Annual Conference on Innovative Applications of Artificial Intelligence (IAAI), Vancouver, British Columbia, Canada, July 22-26 (2007)

16. Jensen, C.S., Lin, D., Ooi, B.C.: Query and update efficient b+tree-based indexing of moving objects. In: Proceedings of Very Large Databases (VLDB), pp. 768–779 (2004)
17. Klein, D., Manning, C.D.: Accurate unlexicalized parsing. In: Proceedings of the 41st Meeting of the Association for Computational Linguistics, pp. 423–430 (2003)
18. Li, Y., Yang, H., Jagadish, H.V.: Nalix: an interactive natural language interface for querying xml. In: SIGMOD Conference, pp. 900–902 (2005)
19. Majithia, S., Shields, M.S., Taylor, I.J., Wang, I.: Triana: A Graphical Web Service Composition and Execution Toolkit. In: Proceedings of the IEEE International Conference on Web Services (ICWS 2004), pp. 514–524. IEEE Computer Society, Los Alamitos (2004)
20. Manola, F., Miller, E.: Resource description framework (rdf) primer. w3c recommendation (2004)
21. No, J., Thakur, R., Choudhary, A.: Integrating parallel file i/o and database support for high-performance scientific data management. In: Supercomputing 2000: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM), Washington, DC, USA, p. 57. IEEE Computer Society, Los Alamitos (2000)
22. Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M.R., Wipat, A., Li, P.: Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics 20(17), 3045–3054 (2004)
23. Ponnekanti, S.R., Fox, A.: Sword: A developer toolkit for web service composition. In: WWW 2002: Proceedings of the 11th international conference on World Wide Web (2002)
24. Sheth, A., Larson, J.: Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases. ACM Computing Surveys 22(3), 183–236 (1990)
25. Shklar, L., Sheth, A., Kashyap, V., Shah, K.: InfoHarness: Use of Automatically Generated Metadata for Search and Retrieval of Heterogeneous Information. In: Iivari, J., Rossi, M., Lyytinen, K. (eds.) CAiSE 1995. LNCS, vol. 932. Springer, Heidelberg (1995)
26. Singh, G., Bharathi, S., Chervenak, A., Deelman, E., Kesselman, C., Manohar, M., Patil, S., Pearlman, L.: A metadata catalog service for data intensive applications. In: SC 2003: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing, p. 33. IEEE Computer Society, Washington (2003)
27. Traverso, P., Pistore, M.: Automated composition of semantic web services into executable processes. In: 3rd International Semantic Web Conference (2004)
28. Tuchinda, R., Thakkar, S., Gil, A., Deelman, E.: Artemis: Integrating scientific data on the grid. In: Proceedings of the 16th Conference on Innovative Applications of Artificial Intelligence (IAAI), pp. 25–29 (2004)
29. Wieczorek, M., Prodan, R., Fahringer, T.: Scheduling of scientific workflows in the askalon grid environment. SIGMOD Rec. 34(3), 56–62 (2005)
30. Web services business process execution language (wsbpel) 2.0, oasis standard
31. Wu, D., Sirin, E., Hendler, J., Nau, D., Parsia, B.: Automatic web services composition using shop2. In: ICAPS 2003: International Conference on Automated Planning and Scheduling (2003)