Semantic Metadata Generation for Large Scientific Workflows

Jihie Kim[1], Yolanda Gil[1], and Varun Ratnakar [1],

[1] Information Sciences Institute, University of Southern California
4676 Admiralty Way, Marina del Rey CA 90292, United States
{jihie, gil, varunr}@isi.edu

Abstract. In recent years, workflows have been increasingly used in scientific applications. This paper presents novel metadata reasoning capabilities that we have developed to support the creation of large workflows. They include 1) use of semantic web technologies in handling metadata constraints on file collections and nested file collections, 2) propagation and validation of metadata constraints from inputs to outputs in a workflow component, and through the links among components in a workflow, and 3) sub-workflows that generate metadata needed for workflow creation. We show how we used these capabilities to support the creation of large workflows in an earthquake science application.

Keywords: metadata reasoning, workflow generation, grid workflows.

1 Introduction

Scientists have growing needs to use computational workflows that address different aspects of the phenomenon under study [14, 5, 2, 28]. In recent years, uses of large workflows have been significantly increased. Often they adopt grid-based environments that enable efficient execution of large workflows by making use of distributed shared resources [26,17]. In such cases, computations in scientific workflows are represented as grid jobs that describe components used, input files required, and output files that will be produced in distributed environment [4].

Metadata describe the data products used and generated by workflow components. Semantic web techniques have been applied for metadata reasoning on workflows such as validation of input parameters based on provenance data using component semantics [27], representing and managing dependencies between data products [15], helping scientists relate and annotate data and services through ontology-based generation and management of provenance data [29], etc. However, most of the existing metadata reasoning approaches focus on analyses of provenance data that are created from execution [22] rather than generation of input and output file descriptions needed in the workflow before execution.

The metadata reasoning capabilities of existing systems focus on files and simple collections and cannot effectively handle constraints on nested collections. Existing checks on files are limited to validation of inputs for individual components.

However, often there are global constraints on inputs and outputs of multiple components, and the workflow should be validated against such constraints in order to prevent execution of invalid workflows and wasting of expensive computations. Unnecessary execution of individual components or multiple components in the given workflow should be detected and avoided when datasets that are equivalent to the ones to be produced exist.

The creation of such large workflows requires several metadata reasoning capabilities:

- Keeping track of constraints on datasets used (i.e. files and file collections), including global constraints among multiple components as well as local constraints within individual components.
- Identifying datasets that are used and produced by the workflow efficiently.
- Detecting equivalent datasets and prevent unnecessary execution of workflow parts when datasets already exist.
- Managing large datasets and their provenance.

This paper presents novel metadata reasoning capabilities that we have developed to support the creation of large workflows. They include 1) use of semantic web technologies in handling metadata constraints on file collections and nested file collections, 2) propagation and validation of metadata constraints from inputs to outputs in a workflow component, and through the links among components in a workflow, and 3) sub-workflows that generate metadata needed for workflow creation. We show how we used these capabilities to support the creation of large workflows in an earthquake science application.

2 Motivation

A computational workflow is a set of executable programs (called *components*) that are introduced and linked together to pass data products to each other. The purpose of a computational workflow is to produce a desired end result from the combined computation of the programs. We will call a computational workflow as a *workflow* in this paper for brevity. Whereas a workflow represents a flow of data products among executable components, a *workflow template* is an abstract specification of a workflow, with a set of *nodes* and *links* where each node is a placeholder for a *component* or *component collections* (for iterative execution of the program over file collections), and each link represents how the input and output parameters are connected. For example, Figure 1-(a) shows a template that has been used by earthquake scientists in SCEC (Southern California Earthquake Center) in Fall 2005. The template has two nodes (seismogram generation and calculation of spectral accelerations), each one containing a component collection. The workflow created from the template is shown in Figure 1-(b). This workflow has been used in estimating hazard level of a site with respect to spectral acceleration caused by ruptures and their variations over time.

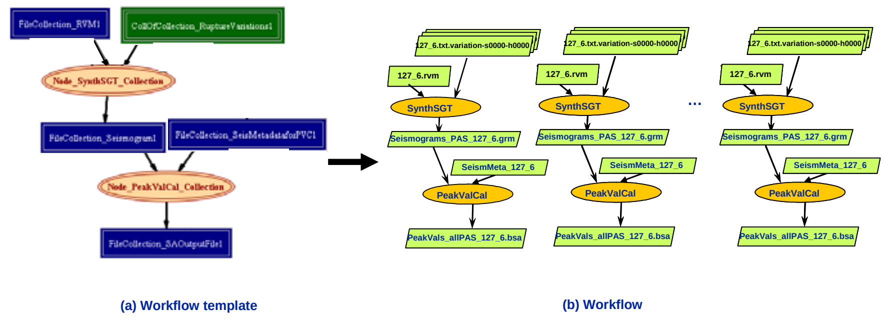

FileCollection_RVM1
CollOfCollection_RuptureVariations1
Node_SynthSGT_Collection
FileCollection_Seismogram1
FileCollection_SeisMetadataforPVC1
Node_PeakValCal_Collection
FileCollection_SAOutputFile1
127_6.txt.variation-s0000-h0000
127_6.rvm
SynthSGT
Seismograms_PAS_127_6.grm
SeismMeta_127_6
PeakValCal
PeakVals_allPAS_127_6.bsa
...
(a) Workflow template
(b) Workflow

Figure 1: Workflow creation for seismic hazard analysis in Fall 2005.

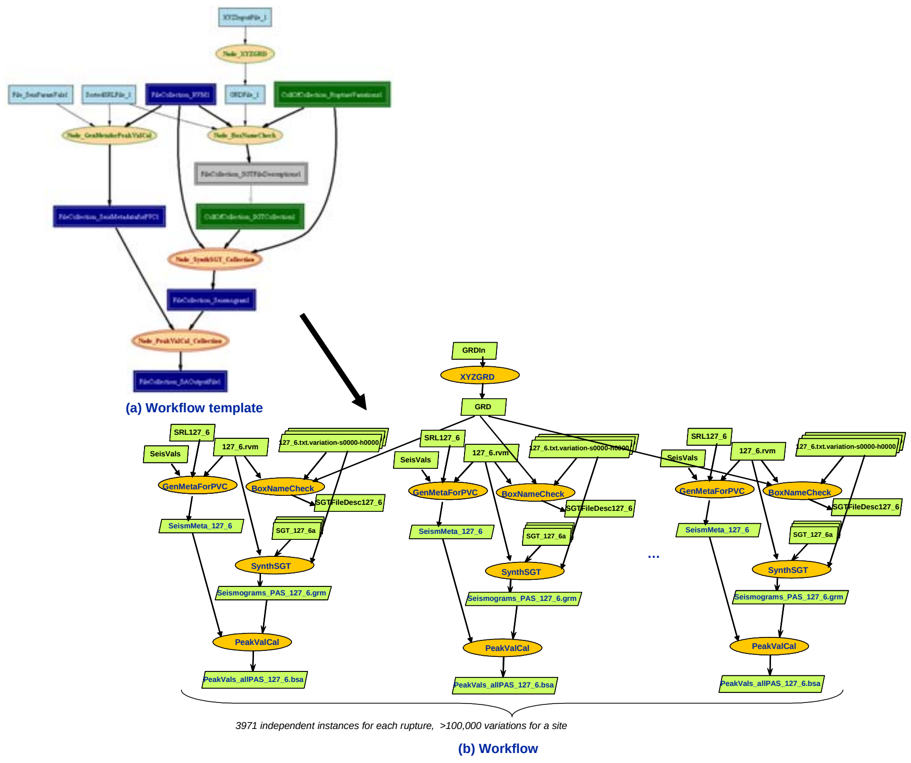


Figure 2: Workflow creation for seismic hazard analysis in Spring 2006.

The workflow was generated from manually created scripts that specify how to bind files to input parameters of the components and what are the expected output file names. An important feature of the workflow is that their data products are stored in files, often organized in directory structures that reflect the structure of the workflows. The names of the files and the directories follow conventions to encode metadata information in the names such as the creation date or the relative area covered by the analysis. Therefore, the scripts that generate the workflow must orchestrate the creation of very particular data identifiers, namely file names that comply with those conventions and are instantiated to the appropriate constants. For example, a file containing the points for a hazard curve would be named using the rupture id and the fault id that were used in the simulation of the wave, as well as the lat-long of the location for the curve.

The script included calls to functions or other scripts that generate information needed by the workflow (e.g. seismic parameter values). These manual 'seam' steps were not a part of the workflow. There were constraints on the files and the collections used by the workflow, but most of the checks were done by hand.

Figure 2-(a) shows an extension in the template in Spring 2006 that includes more nodes and links. This extension was needed to include strain green tensors (SGTs) as additional data input for seismogram generation. As the workflow template and descriptions of components become more complex, the script based approach became infeasible. First of all, there are more manual seam steps to handle. For example, since the names of the SGT files that should be used in the workflow are unknown, the function that generate appropriate SGT file names should be executed beforehand. Validation of the workflow requires more checks. For example, now we need to check whether the SGTs use in generating seismogram are consistent with the rupture variations used for calculating peak values. If the seismogram generation step uses ruptures for Pasadena and their corresponding SGTs but the peak value calculation step uses a rupture variation map for LA, the execution of the workflow will fail. When there exists a dataset that is equivalent to the expected output from executions of some components (e.g. SGT name datasets for Pasadena already exist), scientists could not easily identify them.

In summary, generation of large workflows for this type of applications requires flexibility in adding or changing components to the template, systematic identification of files that are needed and generated by the workflow, incorporation of manual 'seam' steps into the workflow (making them a part of the workflow), and automatic validation of files and collections that are used in the workflow.

3 Approach

In developing new metadata reasoning capabilities for workflow creation, we use a workflow creation framework called Wings [6]. Wings takes a workflow template and initial input file descriptions, and creates an abstract grid workflow called DAX (DAG XML description). A DAX is transformed into an executable concrete workflow through a mapping that assigns available grid resources for execution by Pegasus [4]. Wings uses OWL-DL for representing files and collections, components, workflow templates, and workflows [6]. Currently Jena supports the reasoning.

In this work, Wings was extended to support metadata reasoning and generation. In order to support the above metadata reasoning capabilities, we have developed an approach for representing metadata constraints on files and collections, and a supporting metadata reasoner. Figure 3 highlights the parts that support metadata reasoning capabilities. We are going to describe each in the following subsections.

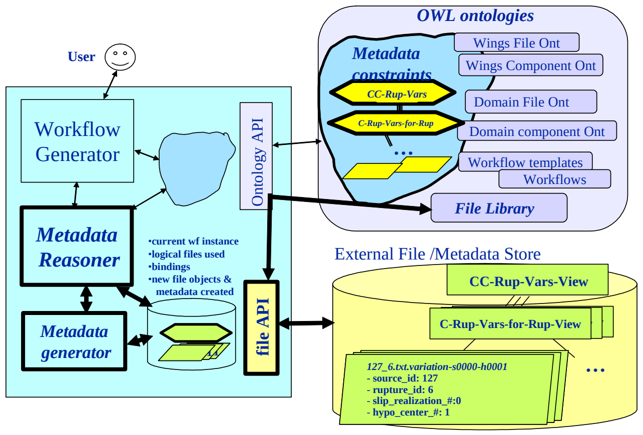
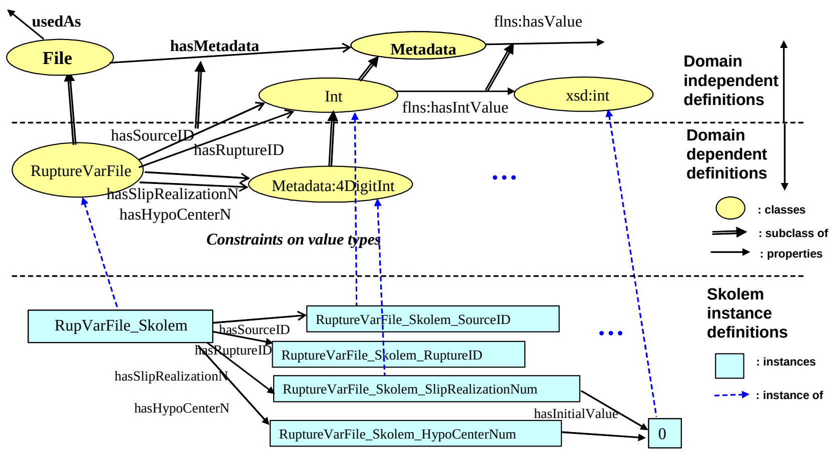


Figure 3: Metadata reasoning for workflow creation

3.1 Representing metadata constraints

In creating workflows, the system needs to keep track of constraints on individual files, constraints on collections and their elements, constraints on inputs and outputs of each component, and global constraints among multiple components.

3.1.1 Metadata constraints on individual files



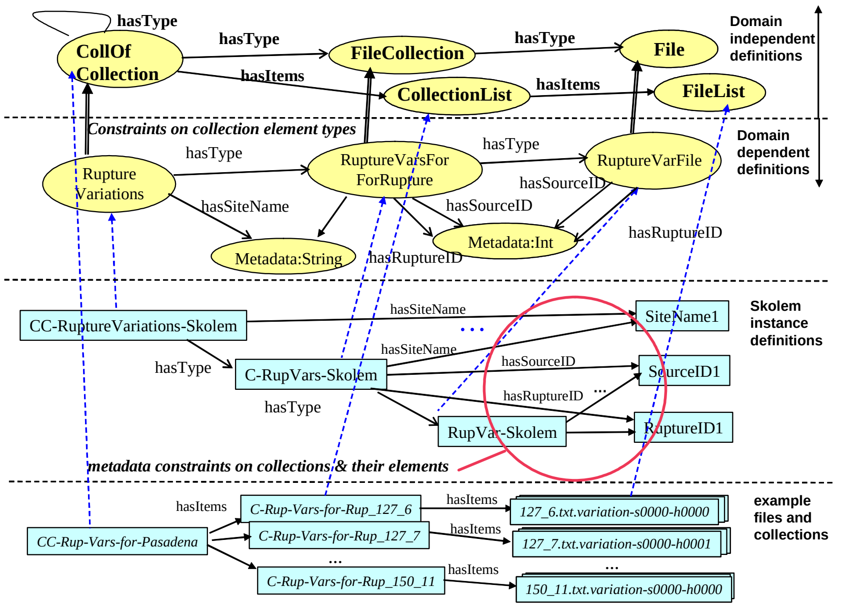
```
<RuptureVariationFile rdf:ID="RuptureVarFile_Skolem"> <rdf:type rdf:resource="&flns;FileSkolem"/>
    <hasRuptureID rdf:resource="#RuptureVarFile_Skolem_RuptureID"/>
    <hasSlipRealizationNumber rdf:resource="#RuptureVarFile_Skolem_SlipRealizationNum"/>
…</RuptureVariationFile>
<flns:Int rdf:ID="RuptureVariationFile_Skolem_RuptureID"/>
<FourDigitInt rdf:ID="RuptureVarFile_Skolem_SlipRealizationNum"> <flns:hasInitialValue
     rdf:datatype="&xsd;int">0</flns:hasInitialValue> …</FourDigitInt>
…
```

flns contains domain independent definitions on files and collections
scecflns contains domain dependent definitions on files and collections

Figure 4: Metadata constraints on individual files

Each file class can have one or more metadata properties associated with it. In representing metadata constraints of a file class, we use a *skolem* instance (e.g., RupVarFile_Skolem) that represents prototypical instances of the class. The metadata can describe what the file contains, how it was generated, how it can be used, etc. For example, a rupture variation file can have Ruptupre ID, SourceID, SlipRelaizationN, and HypoCenterN that represent what it contains. Each metadata property has value ranges and can have some initial values. Other constraints such as how to derive filenames from metadata can be represented using the skolem instance. The actual metadata property values of file instances can be used in checking constraints on input and output files/collections used in the workflow, as described below.

3.1.2 Handling constraints on nested collections

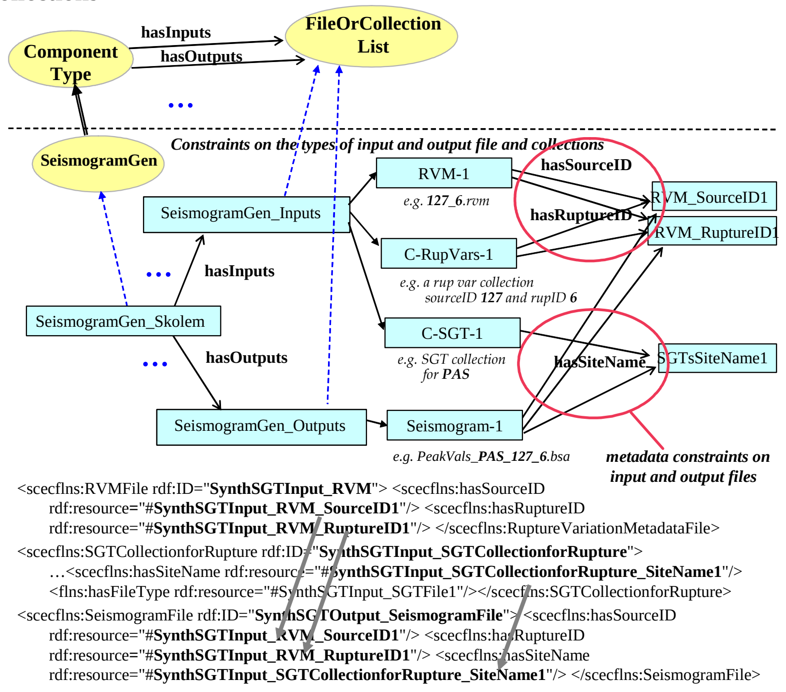


```
<owl:Class rdf:ID="CollOfCollection"> <rdfs:subClassOf rdf:resource="#Collection"/> </owl:Class>
<owl:Class rdf:ID="FileCollection"> <rdfs:subClassOf rdf:resource="#Collection"/> </owl:Class>
 <owl:Class rdf:ID="RuptureVariations"> <rdfs:subClassOf rdf:resource="&flns;CollOfCollection"/>
     <rdfs:subClassOf> <owl:Restriction> <owl:onProperty rdf:resource="&flns;hasType"/> <owl:allValuesFrom
     rdf:resource="#RuptureVariationsforRupture"/> </owl:Restriction> </rdfs:subClassOf> </owl:Class>
<owl:Class rdf:ID="RuptureVariationsforRupture"> <rdfs:subClassOf> <owl:Restriction> <owl:onProperty
     rdf:resource="&flns;hasType"/> <owl:allValuesFrom rdf:resource="#RuptureVariationFile"/>
     </owl:Restriction> </rdfs:subClassOf> <rdfs:subClassOf rdf:resource="&flns;FileCollection"/> </owl:Class>
<scecflns:RuptureVariationsforRupture rdf:ID="BNCI_RuptureVariationsforRupture"> <scecflns:hasSourceID
     rdf:resource="#BNCI_SourceID"/> <flns:hasFileType rdf:resource="#BNCI_RuptureVariationFile"/>
     <scecflns:hasRuptureID rdf:resource="#BNCI_RuptureID"/> </scecflns:RuptureVariationsforRupture>
<scecflns:RuptureVariationFile rdf:ID="BNCI_RuptureVariationFile">
     <scecflns:hasSourceID rdf:resource="#BNCI_SourceID"/>
     <scecflns:hasRuptureID rdf:resource="#BNCI_RuptureID"/> </scecflns:RuptureVariationFile>
```

Figure 5: Collections of file collections and their metadata constraints.

In general, for a given site (e.g. Pasadena in California), more than one ruptures are used in performing the hazard analysis. According to rupture dynamics of earthquakes that depend on hypocenter and slip values, each temporal variation of the stress is described in a rupture variation file. That is, rupture variation files for a site is a collection of file collections. In our ontology, the concept *collection* represents both simple file collections and nested collections. Each collection should specify the type for the collection element using the 'hasType' property. There can be constraints between a collection and its elements. For example, for a rupture variation collection for a rupture, the SourceID and the RuptureID of individual rupture variation file should be the same as the rupture's SourceID and RuptureID. That is, if the rupture variation collection for a rupture has SourceID 127 and RuptureID 6, each element (a rupture variation file) should have SourceID 127 and RuptureID 6. Figure 5 shows how these constraints on collections and nested collections are represented with skolem instances.

3.1.3 Constraints on components: constraints on input and output files and collections



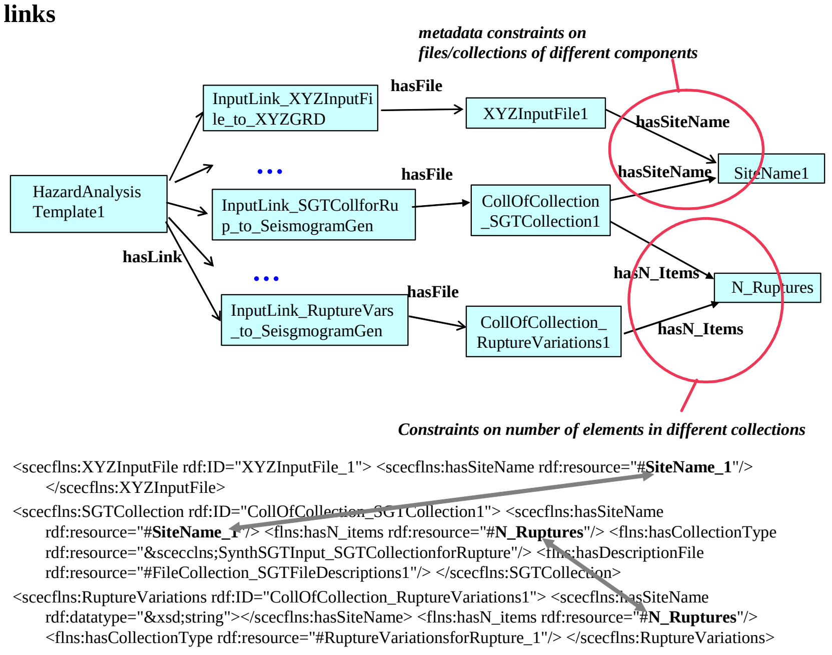
```
<scecflns:RVMFile rdf:ID="SynthSGTInput_RVM"> <scecflns:hasSourceID
      rdf:resource="#SynthSGTInput_RVM_SourceID1"/> <scecflns:hasRuptureID
      rdf:resource="#SynthSGTInput_RVM_RuptureID1"/> </scecflns:RuptureVariationMetadataFile>
<scecflns:SGTCollectionforRupture rdf:ID="SynthSGTInput_SGTCollectionforRupture">
      …<scecflns:hasSiteName rdf:resource="#SynthSGTInput_SGTCollectionforRupture_SiteName1"/>
      <flns:hasFileType rdf:resource="#SynthSGTInput_SGTFile1"/></scecflns:SGTCollectionforRupture>
<scecflns:SeismogramFile rdf:ID="SynthSGTOutput_SeismogramFile"> <scecflns:hasSourceID
      rdf:resource="#SynthSGTInput_RVM_SourceID1"/> <scecflns:hasRuptureID
      rdf:resource="#SynthSGTInput_RVM_RuptureID1"/> <scecflns:hasSiteName
      rdf:resource="#SynthSGTInput_SGTCollectionforRupture_SiteName1"/> </scecflns:SeismogramFile>
```

Figure 6: constraints on metadata properties of input/output files or collections.

Each workflow component is described in terms of its input and output data types. In Figure 6, the SeismogramGen component has three inputs: an RVM (rupture variation map) file, a rupture variation collection, and a SGT file collection. Each RVM file has a SourceID and a RuptureID of the rupture that it represents. In order to create valid results, their values should be the same as the RuptureID and SourceID of the input rupture variation collection. The input SGT collection should have a site name associated with it. Given these inputs, the SeismogramGen component produces a seismogram file.

The metadata for the generated seismogram file depends on the metadata of the inputs. In the above example, the site name of the SGT collection (PAS), and the SourceID and RuptureID of the RVM file (127 and 6) is propagated to corresponding metadata properties of the output seismogram file. The procedure for metadata validation and propagation during workflow creation is described in Section 3.2.

3.1.4 Global constraints on templates: constraints among different nodes and links


metadata constraints on files/collections of different components
InputLink_XYZInputFile_to_XYZGRD
hasFile
XYZInputFile1
hasSiteName
hasSiteName
SiteName1
HazardAnalysis Template1
• • •
hasFile
InputLink_SGTCollforRup_to_SeismogramGen
CollOfCollection_SGTCollection1
hasLink
• • •
hasN_Items
N_Ruptures
hasFile
InputLink_RuptureVars_to_SeisgmogramGen
CollOfCollection_RuptureVariations1
hasN_Items
Constraints on number of elements in different collections

```
<scecflns:XYZInputFile rdf:ID="XYZInputFile_1"> <scecflns:hasSiteName rdf:resource="#SiteName_1"/>
      </scecflns:XYZInputFile>
<scecflns:SGTCollection rdf:ID="CollOfCollection_SGTCollection1"> <scecflns:hasSiteName
      rdf:resource="#SiteName_1"/> <flns:hasN_items rdf:resource="#N_Ruptures"/> <flns:hasCollectionType
      rdf:resource="&sceccIns;SynthSGTInput_SGTCollectionforRupture"/> <flns:hasDescriptionFile
      rdf:resource="#FileCollection_SGTFileDescriptions1"/> </scecflns:SGTCollection>
<scecflns:RuptureVariations rdf:ID="CollOfCollection_RuptureVariations1"> <scecflns:hasSiteName
      rdf:datatype="&xsd;string"></scecflns:hasSiteName> <flns:hasN_items rdf:resource="#N_Ruptures"/>
      <flns:hasCollectionType rdf:resource="#RuptureVariationsforRupture_1"/> </scecflns:RuptureVariations>
```

Figure 7: Global constraints on metadata properties among files and collections used by different components in a template

There are additional validation checks that should be made in order to create a valid workflow. First of all, the components in the template should use seismic data for the same site (e.g. PAS) in performing hazard analysis. In Figure 7, the site name of the XYZinput file that is used in generating a mesh for simulation should be the same as the site name of the SGT collection of collections. (We also use a isSameAs property in representing equalities of metadata.) In addition, the components should use the same number of ruptures throughout the workflow. For example, the number of elements in a collection of collection rupture variations indicates the number of ruptures used in modeling the site. This number (i.e. the number of ruptures) should be the same as the number of elements (SGT collections) in the collection of collection SGTs that are used. If the specific number of ruptures are known, the value can be given for the N_Ruptures using the flns:hasValue property. Figure 7 shows the current representations that are used. In representing these global constraints, we make use of *link skolems*. Each link skolem is a placeholder for a file or collection that is bound to the input and output parameters of the link during workflow creation. When more than one link skolems in a template share the same metadata objects, when the bindings for the links are created, their corresponding metadata values should be the same. These constraints are used by metadata reasoner in creating consistent and correct workflows. The details of metadata based validation are described below.

3.2 Metadata propagation and validation

Table 1: Steps for propagating metadata and checking constraints during workflow creation.

```
Bind&ValidateWorkflow (WorkflowTemplate wt, InputLinks ILinks)
  Assign ILinks to LinksToProcess.
  While LinksToProcess is not empty
    Remove one from LinksToProcess and assign it to L1.
    Let F1 be the link skolem for binding files or collections to L1.
    If metadata for F1 should be generated from an execution of a component
        if the execution results are not available, continue. ;; exclude this link in the sub-workflow
    If any metadata of F1 depends on a link L2 that is not bound yet, mark L1 as a dependent of L2
        and continue.
    If L1 is an input link, get metadata of the file from the user or a file server
        check consistencies with links that L1 depends on ;; consistency check
        check consistencies with existing bindings based on template-level constraints
                                                               ;; consistency check
        If any metadata are inconsistent, report inconsistency and return.
        Bind file/collection name and metadata to F1.
        If the file type for F1 is a collection, recursively get the metadata of its elements
    Else (L1 is InOutLink or OuputLink) generate file names and metadata base on the definition of
        the depending links. ;; metadata propagation
    For each link L2 that is dependent on l1, if all the links that L2 is depending on are bound,
        put L2 in LinksToProcess.
    If L1 is an output link, continue.
    Else (L1 is InputLink or InOutLink) if all the inputs to the destination node (i.e. the component
        that L1 provides an input to) have been bound, add all the OutputLinks and InOutLinks from
        the destination node to the LinksToProcess.
```

Table 1 shows the Bind&ValidateWorkflow procedure for propagating metadata constraints and validating workflows created using metadata constraints. The procedure traverses links in the workflow template and generate consistent bindings for link skolems. There are three classes of links: InputLink, InOutLink, and OutputLink. An InputLink is a link from an initial input file or collection to a node. Each InOutLink represents a connection from an output parameter of a node to an input parameter of another node. An OutputLink represents an end result from a node. The procedure specifies how the system starts with the input links of a template, identifies dependencies among the links based on definitions of metadata constraints, binds link skolems to files or collections, propagates and checks constraints of the bindings based on metadata constraints, and traverses the next unbound links based on the dependencies.

A link l1 is dependent on l2 if some of the metadata of l1 can be filled in based on some metadata of l2. For example, in Figure 6, the output of SeismogramGen step depends on the RVM file and the SGT collection. The input link for a rupture variation collection depends on the input link for an RVM, if the SourceID and the RuptureID of the rupture variations are derived from the values in the RVM file. We assume that there are no cyclic dependencies in the definition of metadata constraints.

The file names and the metadata for initial input files or collections can be given from the user or existing file library (in OWL) through a file API. The metadata of the initial inputs can also be retrieved from other external file stores using the same API. Currently we use a web repository, but we are exploring uses of grid file repositories such as MCS (metadata catalog service) [23]. The italicized steps handle sub-workflows, which are explained in the next section. The Bind&ValidateWorkflow procedure can be used in two different modes: an interactive mode where the user can enter input file/collection names and metadata in an interactive fashion, or a file server mode where the metadata for input files are filled in through file sever calls.

3.3 Sub-workflows for generating metadata needed for workflow creation

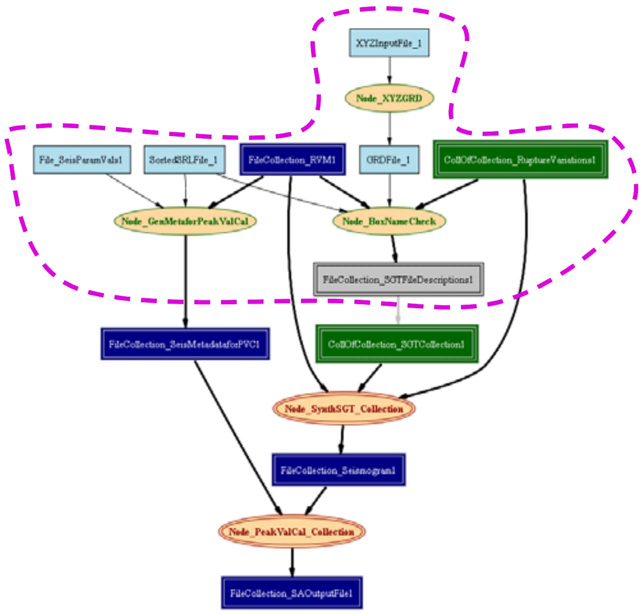

XYZInputFile_1
Node_XYZGRD
File_SeisParamVals1
SortedSRLFile_1
FileCollection_RVM1
GRDFile_1
CollOfCollection_RuptureVariations1
Node_GenMetaforPeakValCal
Node_BoxNameCheck
FileCollection_SGTFileDescriptions1
FileCollection_SeisMetadataforPVC1
CollOfCollection_SGTCollection1
Node_SynthSGT_Collection
FileCollection_Seismogram1
Node_PeakValCal_Collection
FileCollection_SAOutputFile1

Figure 8: Identifying and executing sub-workflows for full workflow creation.

In order to minimize uses of manual seam steps in creating workflows, including calls to functions that generate file names, we have created new workflow components that model such manual steps. For example, in Figure 8 (an enlargement of Figure 2-(a)), individual files in CollOfCollection_SGTCollection1 are unknown initially and the file names need in the workflow should be generated by executing the BoxNameCheck component. Previously, the execution of BoxNameCheck was done manually. We represent such components as workflow components, and link them to the depending components inputs or outputs (e.g. SGT files needed by SynthSGT) in the workflow template.

In generating grid workflows, for each execution of a component, the names of the inputs and output files for the component should be specified. That is, what data are published, and what data are staged in and out of the computation should be known before execution. Often, names or descriptions of some of these files are not given initially, and their names should be computationally generated.

As shown in Table 1, our Bind&ValidateWorkflow procedure checks these dependencies, and generates a 'sub'-workflow that includes only the parts that can be instantiated with the currently available data. For the template in Figure 8, a sub-workflow with bindings for input and output links of the three components (XYZGRD, GenMetaForPeakValCal and BoxNameCheck), highlighted with dotted lines, is generated. The resulting sub-workflow is mapped to grid resources through Pegasus [5] and executed in a grid environment. The execution of a sub-workflow provides results for dependent components, such as file names needed for dependent components inputs or outputs. The metadata for these new file names are generated and added to our file repository by the metadata generator (shown in Figure 3) so that they can be used in creating an expanded workflow. Currently making the results available in the file repository is done in a semi-automatic fashion, but we are investigating client-server style interaction between workflow creation and execution, as described below. The creation and execution of sub-workflows can be performed repeatedly until the complete workflow is generated. The above workflow template needs only one iteration of sub-workflow creation and execution, before the full workflow is created.

4 Results

Table 2: Number of files and OWL instances created during workflow creation

	Workflow creation time	Number of file instances created for the workflow	Number of OWL individuals created
A sub workflow for hazard analysis	7 minutes, 59 seconds	15,888	322,473
A full workflow for hazard analysis	22 minutes, 52 seconds	117,379	2,001,972

The above metadata reasoning capabilities are used in creating workflows for earthquake hazard analysis. In creating a workflow for an LA site with the template in Figure 8, there were about 3,971 ruptures and 97,228 variations of ruptures to take into account. As the number of files and file collections become large, many OWL objects that represent file and collections and their metadata should be created and queried. The number of files in the workflow we have represented was 117,379, as shown in Table 2. The number of OWL individuals created was over two million. (We excluded the anonymous individuals that are created as a by-product of rdf:list in the count, so the actual number is larger.) For the full workflow, the DAX included 7,945 jobs. Large workflows pose challenges on computational resources (CPUs and memory) used during workflow creation. Currently it takes about 8 minutes to create a sub workflow and about 23 minutes to generate the full workflow on a Pentium 4 3.0GHz with 1GB of RAM.

In order to efficiently perform the required metadata reasoning with many objects, we split a workflow into multiple independent workflow parts and create them separately. In splitting, we make use of metadata properties that can divide collections into independent sub-collections. For example, separate sub-trees in Figure 2-(b) can be independently generated. We currently use the SourceID to split rupture file collections into sub-collections. Other collections such as rupture variation collections are divided using the same set of metadata properties. Currently we select such metadata properties by hand, but we are investigating an automatic approach that takes into account distributions of file collections. The independent workflow parts are accumulated in the workflow generator and are automatically merged in the end, creating a complete workflow.

We make use of the existing Wings capability for saving newly created file objects in OWL. Using the same collection splitting approach described above, we can store the resulting files and collections into separate file library entities. The objects can be selectively loaded and used in creation of new workflows.

Equivalent files or collections can be identified using metadata, which enables detection of unnecessary execution of components or workflow parts that will produce equivalent datasets by Pegasus [4].

5 Related work

Semantic web techniques have been used in supporting many e-science workflow systems [10, 7]. Applications include semantic description of web services, resource discovery, data management, composition of workflow templates [18, 21, 24, 1], etc. Our work complements existing work by supporting creation of large workflows needed for data and/or compute intensive scientific computations.

Recently various data management and provenance techniques have been developed for e-Science applications [22,8]. Most of the existing work focuses on pedigree or lineage metadata that describes the data resources and the processes used in generating data products. These provenance metadata is often used in qualifying data products and supporting data management and reuse. Our current work focuses metadata on data content that support identification of consistent file collections used in workflow creation. The newly created metadata on file content can be used in combination of other provenance metadata in supporting file reuse. Our work extends existing approaches for validating workflows in that we take into account constraints on nested collections and global constraints among multiple components as well as constraints on inputs within individual components [27,15]. Another difference is that we make use of metadata in generating valid workflows before execution instead of validating already executed workflows with provenance data, enabling detection of unnecessary jobs before execution.

As large OWL applications increase over the years, efficiency issues of OWL reasoning capabilities have been investigated and new techniques have been proposed in recent years [9,11,16]. In addition to using a database backend for our metadata reasoner, we are planning to explore options for more efficient OWL and RDF reasoning approaches.

6 Conclusions and future work

We presented a semantic metadata generation and reasoning approach that supports creation of large workflows. Given the metadata of initial input files, the system propagates metadata constraints from the inputs to the outputs, and through the links among the components during workflow creation. Both global constraints among multiple components and local constraints are used for workflow validation. The files that will be produced from workflow execution as well as the input files are identified during the metadata propagation and validation process. Some of the metadata are generated through creation and execution of sub-workflows when the metadata need to be computationally generated. Because we are able to identify data collections and their properties before the workflow is executed, we can detect whether the data has been generated before by querying an existing data repository. This is important for optimizing execution performance: If some intermediate data product already exists then there is no need to re-execute the portion of the workflow that produces it. We also use the metadata in managing large collections and their provenance.

We are currently working on extensions of the workflow template shown in Figure 2-(a) and they will use more datasets for seismic analysis of different sites in Southern California. In order to further improve the efficiency of the workflow creation and metadata reasoning, we are considering several extensions to our system. First of all, we are working on a database backend for storing metadata for many files and collections. Currently we can store them in multiple owl files, but we are planning to explore uses of MCS that can store metadata of data products (such as files) published on the Grid [23]. With this approach, when there are new files and metadata added to MCS by a different client, we will be able to use them in creating new workflows. In order to perform iterative sub-workflow generation and execution more efficiently, we are investigating a client-server style approach where our system can call a workflow execution server with a newly generated sub-workflow, and the execution results can be notified to our system (a client). The newly generated metadata during workflow creation can be used in combination with other metadata for data provenance applications. For example, the metadata can tell whether the two files (or collections) contain the same kind of information, even when they are generated from different workflows. We are exploring various uses of metadata in relating datasets used in scientific workflows.

References

1. I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, S. Mock, 2004. Kepler: Towards a Grid-Enabled System for Scientific Workflows, In the Workflow in Grid Systems Workshop in GGF10 - The Tenth Global Grid Forum, Berlin, Germany, March 2004.
2. M. Campobasso and M. Giles, 2004.Stabilization of a Linear Flow Solver for Turbomachinery Aeroelasticity Using Recursive Projection Method,*AIAA Journal*,Vol.42, no.9.
3. D.,Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor and I. Wang, Programming Scientific and Distributed Workflow with Triana Services. In Grid Workflow 2004 Special Issue of Concurrency and Computation: Practice and Experience.
4. E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M. Su, K. Vahi, M. Livny 2004., Pegasus : Mapping Scientific Workflows onto the Grid , Across Grids Conference.

5. E. Deelman, J. Blythe, Y. Gil, and C. Kesselman, 2003. Workflow Management in GriPhyN. The Grid ResourceManagement, Kluwer.
6. Y. Gil, V. Ratnakar, E. Deelman, M. Spraragen, and J. Kim, 2006. Wings for Pegasus: A Semantic Approach to Creating Very Large Scientific Workflows, an internal project report, http://www.isi.edu/ikcap/scec/papers/Wings-for-Pegasus.pdf.
7. C. Goble, 2005. Using the Semantic Web for e-Science: Inspiration, Incubation, Irritation Lecture Notes in Computer Science 3729:1-3.
8. C. Goble, 2002. "Position Statement: Musings on Provenance, Workflow and (Semantic Web) Annotations for Bioinformatics," in Workshop on Data Derivation and Provenance.
9. Y. Guo, Z. Pan, and J. Heflin, 2004. An Evaluation of Knowledge Base Systems for Large OWL Datasets, Third International Semantic Web Conference.
10. J. Hendler, 2003. Science and the Semantic Web Science 299: 520-521.
11. U. Hustadt, B. Motik, U. Sattler, 2005. Data Complexity of Reasoning in Very Expressive Description Logics. Proc. of the 19th International Joint Conference on Artificial Intelligence.
12. J. Kim, M. Spraragen, and Y. Gil, 2004. An Intelligent Assistant for Interactive Workflow Composition, Proceedings of the International Conference on Intelligent User Interfaces, 2004.
13. P. Kovatch, 2004. TeraGrid Software Strategy: E Pluribus Unum, Department of Defense Software Tech News, April 2004.
14. P. Maechling, et al., 2005. Simplifying Construction of Complex Workflows for Non-Expert Users of the Southern California Earthquake Center Community Modeling Environment, ACM SIGMOD Record, special issue on Scientific Workflows, Vol.34, Issue 3.
15. J. Myers, C. Pancerella, C. Lansing,, K. Schuchardt, B. Didier, 2003. Multi-scale Science: Supporting Emerging Practice with Semantically-Derived Provenance, Semantic Web Technologies for Searching and Retrieving Scientific Data Workshop, 2003.
16. openRDF, 2006. http://www.openrdf.org/, 2006.
17. Open Science Grid, http://www.opensciencegrid.org/gt4
18. OWL-S, 2006. http://www.daml.org/services/owl-s/, 2006.
19. OWL Web Ontology Language, 2006. http://www.w3.org/TR/owl-features/, 2006.
20. L. Pearlman, Metadata Catalog Service for Data Intensive Applications, SC 2003.
21. M.Sabou, C. Wroe, C. Goble and G. Mishne Learning Domain Ontologies for Web Service Descriptions: an Experiment in Bioinformatics in Proc 17th Intl Conference on World Wide Web.
22. Y. Simmhan and B. Plale and D. Gannon, 2005, A Survey of Data Provenance in e-Science, in SIGMOD Record, vol. 34, 2005, pp. 31-36.
23. G. Singh, S. Bharathi**,** A.Chervenak, E. Deelman, C. Kesselman, M. Manohar, S. Patil, and L. Pearlman, 2003, A Metadata Catalog Service for Data Intensive Applications, SC 2003.
24. E. Sirin, B. Parsia, and J. Hendler, 2004. Filtering and selecting semantic web services with interactive composition techniques. IEEE Intelligent Systems, 19(4):42-49, 2004.
25. K. Sycara, M. Paolucci, A. Ankolekar and N. Srinivasan, 2003. Automated Discovery, Interaction and Composition of Semantic Web services," in Journal of Web Semantics, Volume 1, Issue 1.
26. TeraGrid 2006. NSF Teragrid Project, http://www.teragrid.org/.
27. S. Wong, Miles, S., Fang, W., Groth, P. and Moreau, L. (2005) Validation of E-Science Experiments using a Provenance-based Approach. In Proceedings of 4th UK e-Science All Hands Meeting (AHM), Nottingham.
28. C. Wroe, C. Goble, M. Greenwood, P. Lord, S. Miles, J. Papay, T. Payne, L. Moreau, 2004. Automating Experiments Using Semantic Data on a Bioinformatics Grid in IEEE Intelligent Systems special issue on e-Science Jan/Feb 2004.
29. J. Zhao, C. Goble, R. Stevens and S. Bechhofer, 2004. Semantics of a Networked World: Semantics for Grid Databases, First International IFIP Conference, ICSNW 2004.