Provenance-based Validation of E-Science Experiments

Sylvia C. Wong, Simon Miles, Weijian Fang, Paul Groth and Luc Moreau

School of Electronics and Computer Science,
University of Southampton, UK
{sw2,sm,wf,pg03r,l.moreau} *at* ecs.soton.ac.uk

Abstract. E-Science experiments typically involve many distributed services maintained by different organisations. After an experiment has been executed, it is useful for a scientist to verify that the execution was performed correctly or is compatible with some existing experimental criteria or standards. Scientists may also want to review and verify experiments performed by their colleagues. There are no exsiting frameworks for validating such experiments in today's e-Science systems. Users therefore have to rely on error checking performed by the services, or adopt other ad hoc methods. This paper introduces a platform-independent framework for validating workflow executions. The validation relies on reasoning over the documented *provenance* of experiment results and *semantic descriptions* of services advertised in a registry. This validation process ensures experiments are performed correctly, and thus results generated are meaningful. The framework is tested in a bioinformatics application that performs protein compressibility analysis.

1 Introduction

Very large scale computations are now becoming routinely used as a methodology to undertake scientific research: success stories abound in many domains, including physics (`griphyn.org`), bioinformatics (`mygrid.org.uk`), engineering (`geodise.org`) and geographical sciences (`earthsystemgrid.org`). These large scale computations, which underpin a scientific process usually referred to as *e-Science*, are ideal candidates for use of Grid technology [1].

E-Science experiments are typically formed by invoking multiple services, whose compositions are modelled as workflows [2]. Thus, experimental results are obtained by executing workflows. As part of the scientific process, it is important for scientists to be able to verify the correctness of their own experiments, or to review the correctness of their peers' work. Validation ensures results generated from experiments are meaningful.

Traditionally, program validation has been carried out in two complementary manners. On the one hand, *static verification* analyses program code before it is executed and establishes that the program satisfies some properties. These verifications are extensively researched by the programming language community. Examples include type inference, escape analysis and model checking. They typically depend on the semantics of the programming language being analysed. On the other hand, static verification is complemented by *run-time* checking, which is carried out when the program executes, and verifies that data values satisfy constraints, expressed by either types or assertions.

Such validation methods suffer from limitations in the context of large e-Science experiments, potentially carried out in open environments. First, programs (or workflows) may not be expressed in languages that analysis tools operate on, or may not be directly available because they are exposed as services, hereby preventing static analysis. Second, in general, in open environments, we cannot make the assumption that services always check that their inputs or outputs match their interface specifications (if available at all); furthermore, such interfaces may be under-specified (for instance, many bioinformatics services tend to process and return strings encoding specific biological sequence data); as a result, no guarantee exists that types will be checked dynamically. Third, studies of user practice have shown that rapid development cycles are being adopted by e-Scientists, in which workflows are frequently modified and tuned and scientific models are evolved accordingly. As a result, it is important for scientists to be able to verify that previous experimental results are compatible with recent criteria and requirements. Since these models did not necessarily exist at experiment design or execution time, it is a necessity to perform such validation *after* the experiment has been completed.

The *provenance* of a piece of data denotes the process by which it is produced. Provenance-aware applications are applications that record documentation of their execution so that the provenance of the data they produce can be obtained and reasoned over. In this paper, our thesis is that provenance-based validation of experiments allows us to verify the validity after experiments have been conducted. Specifically, our contributions are: (a) a provenance-based architecture to undertake validation of experiments; (b) the use of semantic reasoning in undertaking validation of experiments; (c) an implementation of the architecture and its deployment in a bioinformatics application in order to support a set of use cases. Our experimentation with the system shows that our approach is tractable and performs efficiently.

The structure of the paper is as follows. Section 2 describes some use cases we have identified that require experiment validation. Section 3 briefly discusses current approaches to e-Science experiment validation and explains why it is necessary to perform validation after an experiment was executed. Section 4 introduces the proposed framework for validation of workflow execution. Section 5 then describes how the architecture can be applied to the use cases introduced in Section 2. In Section 6, we discuss how semantic reasoning is essential in properly establishing the validity of experiments. Section 7 then presents results from an implementation of the validation framework with an e-science application (specifically, the protein compressibility analysis experiment). The paper finishes with discussion in Section 8 and conclusions in Section 9.

2 Use Cases

The motivation for this work comes from real problems found by scientists in their day to day work. Therefore, in this section, we introduce a number of use cases in the bioinformatics domain where it is necessary to perform some form of validation of experiments after they have been completed.

Use Case 1 (Interaction validity, interface level) *A biologist, B, performs an experiment on a protein sequence. One stage of this experiment involves generating a pre-specified number of permutations of that sequence. Later, another biologist, R, judges the experiment results and considers them to be suspicious. R determines that the number of permutations specified was an invalid value, e.g. it was negative.* □

In this example, we consider that the service provider could have specified a restriction for the number of permutations to non-negative integers in the service schema, since the parameter only makes sense for non-negative integers. However, this does not guarantee that the service will validate the data against the schema at run-time. In general, whether validation is carried out at run-time is service specific.

In Use Case 1, B could have entered a negative value for the number of permutations. In this case, the value is incorrect because it does not conform to the restrictions and requirements as specified by the interface document of the service. By validating the experiment using its provenance, R can determine that B entered an invalid value for the number of permutations, and thus the results generated by the experiment were not meaningful.

Use Case 2 (Interaction validity, domain level) *A bioinformatician, B, downloads a file containing sequence data from a remote database. B then processes the sequence using an analysis service. Later, a reviewer, R, suspects that the sequence may have been a nucleotide sequence but processed by a service that can only analyse meaningfully amino acid sequences. R determines whether this was the case.* □

Nucleotides and amino acids are two separate classes of biological sequences, but the symbols used in the syntax of nucleotides are a subset of those used for amino acids. Therefore, it is not always possible to detect which type of sequence is used by superficially examining the data. The service used in Use Case 2 could require an amino acid sequence as its input. If a nucleotide sequence was accidentally used rather than an amino acid sequence, the problem would not be detected at run-time, and the experiment results would not be meaningful.

Given that many bioinformatics services operate on strings, the biological interpretation of a piece of data is information not directly available from interface specification, and cannot be easily derived from the data itself. Typically, such additional description that is useful or of interest to the user has to be made explicit elsewhere. Thus, the interaction in an experiment can be correct according to service interface specifications, but incorrect according to the domain level understanding of the problem.

Use Case 3 (Ontology revision) *A bioinformatician, B, performs an experiment on a sequence downloaded from a remote database. Later, another bioinformatician, D, updates the ontology that classifies sequences stored in the database to correct an error in the previous version. B checks if the experiment is compatible with the new version of the ontology.* □

Ontologies are invaluable in describing domain specific knowledge such as DNA and RNA sequences are subtypes of nucleotide sequences, as illustrated by the Gene

Ontology [3]. If a service advertises that it accepts nucleotide sequences, we can infer that the service can also meaningfully process DNA and RNA sequences.

Similar to Use Case 2, the bioinformatician B in Use Case 3 wants to validate the interactions in the experiment according to their domain-level characterisation (specifically, biological sequence types). However, in this use case, the ontology describing the gene sequences is revised after the experiment has been conducted. Therefore, to ensure results of the experiment are not affected by this error in the ontology, B validates the execution against the revised ontology.

Use Case 4 (Conformance to plan) *A biologist, B, creates a plan for an experiment by defining the type of analysis to perform at each stage of the experiment. B then performs an experiment that is intended to follow the plan. Later another biologist, R, determines whether each operation performed in the experiment fulfilled an intended operation in the plan.* □

In Use Case 4, the plan defined by B is abstract in nature. To verify whether the experiment conformed to the original plan, R examines the *tasks* the services perform. In other words, R is interested in verifying the properties of the services, not the interactions between the services. This is in contrast to the previous use cases, where the validation is performed on the types of the data provided and accepted by the services.

Use Case 5 (Patentability of results) *A biologist, B, performs an experiment. Later, B wishes to patent the results. A reviewer, R, checks that no service used in the experiment has legal restrictions such that the results could not be patented.* □

In Use Case 5, R is interested in attributes such as condition of use, legal constraints and patents. These conditions are (probably) unforeseen by biologist B when he designed and performed the experiment.

3 Current Validation Approaches

Web Services are described by a WSDL interface [4] that specifies the operations they support, the inputs they expect, and the outputs they produce. The inputs and outputs of an operation are part of a message and their structure, referred to as interface type, is commonly specified using XML Schema [5]. In other words, it is the type expected by the transport layer (i.e., SOAP [6]). It is generally (but not always) the role of the service provider to publish interface type definitions.

We augment interface types with further descriptions that characterise additional invariants of interest to the user. For instance, in the previous section, we discussed a characterisation of data in domain-level terms. OWL-S [7] allows for semantic types expressed using the OWL ontology to be added to the service profile. Given that the world is evolving, we consider that several views about an object may co-exist. Hence, it is permitted to associate several semantic types to a given entity: this is the approach adopted by myGrid [8], which also relies on the OWL ontology language to give a classification of biological data. Such descriptions are not restricted to inputs and outputs, but can be annotations to service interfaces that identify the functions they perform or the resources they rely upon. Such information may be provided by the service provider, or by a third party, and published in a registry such as the myGrid/Grimoires registry [9].

In the introduction, we discussed two commonly used forms of validation: static and dynamic. *Static validation* operates on workflow source code. The vast array of static analyses devised by the programming language community is also applicable to workflows, such as type inference, escape analysis, etc. Some analysis were conceived to address problems that are specific to workflows. Examples of these include workflow concurrency analysis [10], graph-based partitioning of workflows [11], and model checking of activity graphs [12]. Yang et al. [13] devise a static analysis to infer workflow quality of service. However, the workflow script may not always be available, or it may be expressed in a language for which we do not have access to a static analyser.

Hence, validation may be performed at run-time. In its simplest form, validation is *service-based*. In Web Services, a validating XML parser verifies all XML documents sent to a service conform to its specified schema. Thus, if all the services used in a workflow employ validating parsers, the workflow execution is guaranteed to satisfy syntactic types required by services. We note however that many XML parsers are non-validating by default, such as Apache Axis (`ws.apache.org/axis`) and JAXB (`java.sun.com/xml/jaxb`), because validation is an expensive operation that affects the performance of web services. Therefore, most XML parsers used by web services simply check if XML documents are well-formed, and if they can be unmarshalled into compiled classes.

Other forms of validation and analysis can take place at run-time. The Pegasus planner is capable of analysing a workflow and re-planning its execution at run-time so as to make use of existing available resources [14]. Policy languages such as KAoS are used to perform semantic reasoning and decide if access can be granted to services as they are being invoked [15].

Service-based validation can only be performed at run-time. However, it is sometimes necessary to validate an experiment after it has been executed. This has been identified in Use Cases 3 and 5. Third parties, such as reviewers and other scientists, may want to verify that the results obtained were computed correctly according to some criteria. These criteria may not be known when the experiment was designed. This is because as science progresses, criteria evolve. Thus, it is important that previously computed results can be verified according to revised sets of criteria.

4 Provenance-based Validation Architecture

We propose a provenance-based approach to workflow validation. The provenance of an experiment contains a record of all service invocations such that the information is sufficient to reproduce the exact experiment. A provenance-based approach lends itself easily to third party validation as scientists can share provenance data with other scientists. Also, as validation criteria evolve, the validation process can be repeated without re-executing the experiment.

Figure 1 explains our proposed provenance-based semantic validation framework. Service providers host services on the Grid and advertise them in a registry. Since we wish to support multi-level descriptions beyond interface types, possibly provided by third parties, the registry provides support for semantic annotations [9]. An interface for metadata publication allows for metadata annotations to services, individual operations (within a service), their inputs and outputs; likewise, a query interface caters for metadata-based discovery.

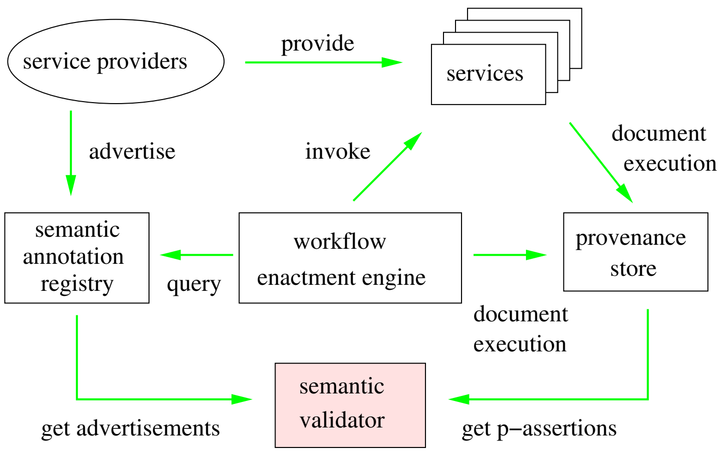

Fig. 1. Provenance-based validation architecture.

Users construct workflows for their experiments. The workflow enactment engine queries the registry for services that provide the tasks requested in the workflow and calls the appropriate services in the correct order. The services and the workflow enactment engine document the execution of the experiment using a provenance store. We refer to the provenance of some experimental result as the documentation of the process that led to that result. Each client and service (collectively, *actors*) in an experiment can assert facts about that experiment, called *p-assertions* (assertions, by an actor, pertaining to provenance). A p-assertion may state either the content of a message sent by one actor to another, an *interaction p-assertion*, or the state of an actor when an interaction took place, an *actor state p-assertion*. Examples of actor state p-assertions range from the workflow that is being executed, to the amount of disk and CPU a service used in a computation [16].

After the experiment is carried out, validation is performed using the algorithm outlined in Figure 2. Validation is done on a per activity basis. In this context, activities are service invocations in the experiment. The list of activities in an experiment is provided by the provenance store. For each activity, a, the validator computes two values for comparison — a requirement on the value of some property, R, and the actual value of that property used in the activity A. The validator then performs semantic reasoning over A and R to see if A fulfils all the requirements specified in R. If A satisfies R, then a is deemed to be valid. An experiment is valid when all activities are proved to be valid.

```
$isValid \leftarrow$ true
for all activities $a$ do
    $(R, A) \leftarrow Compute(a)$
    $isValid \leftarrow isValid \wedge (A$ satisfies $R)$
end for
```

Fig. 2. Algorithm for provenance-based validation. Requirement R and actual value A are calculated using the $Compute$ function shown in Figure 3

Figure 3 explains how requirement R and actual value A are calculated for a given activity a. First, the validator obtains provenance p-assertions for a from the provenance store. Using this provenance information, the validator fetches services' advertisements and semantic annotations from the registry. The user supplies extra information needed for validation, such as the bioinformatics ontology in Use Case 3 and the legal descriptions in Use Case 5.

```
Function: Compute requirement $R$ and actual value $A$
Require: activity $a$
    Retrieve p-assertions from provenance store
    Get advertisements from registry
    Get user supplied information
    $R \leftarrow$ Compute requirements
    $A \leftarrow$ Compute trace
```

Fig. 3. Algorithm to compute requirement R and actual value A.

The type of information to obtain from the provenance store, the registry and the user depends on the actual validation to be performed. Similarly, the semantic reasoning needed to compare requirement R and actual value A also depends on the type of validation. The next section explains how the semantic validator implements the various types of validations identified by the use cases using the algorithms introduced in this section. Section 6 then discusses the semantic reasoning performed.

5 Validation Algorithms for the Use Cases

Figure 3 presented a generic algorithm for computing requirement R and actual value A of an activity by querying the provenance store and the registry. In this section, we apply the algorithm in Figure 3 to the use cases in Section 2.

5.1 Interface level interaction validity

Use Case 1 requires the validation of workflow interactions at the interface level. A workflow is valid if data passed to all activities in the workflow conform to specifications in their WSDL interface documents, defined in XML schema. Specifically, the validator validates input XML documents (actual value A) against the schemas (requirement R). For each activity a, R and A are computed according to Figure 4. The validator queries the provenance store for the service and operation names of activity a. These names are used to obtain the WSDL document for the activity from the registry. The provenance store also provides the validator with a copy of the data passed to the activity in the experiment.

```
Retrieve service/operation names of $a$ from provenance store
$R \longleftarrow$ Get WSDL document for $a$ from registry
$A \longleftarrow$ Retrieve input to $a$ from provenance store
```

Fig. 4. Interface-level interaction validation: computing requirement R and actual value A for activity a.

5.2 Domain level interaction validity

To support Use Cases 2 and 3, we validate all interactions in a workflow execution using domain-level knowledge. For each activity a, we wish to compare the domain-level types of the data expected by the activity (R) with the actual data used (A). The domain-level type of the actual data passed to activity a is derived from the output of preceding operation p. (By preceding, we refer to the service that precedes activity a in terms of data flow, not time). In the simplest case, an interaction is considered domain-level valid if A is either the same type or a subtype of R. Figure 5 summarises how the two values R and A are computed. First, the validator queries the provenance store to obtain the service and operation names of activity a and preceding activity p. With the service and operation names, the validator retrieves the metadata attached to the WSDL message parts from the registry. Specifically, the validator is interested in the metadata for the output message part of operation p, and the metadata for the input message parts of the current operation a. The last piece of information the validator requires is the ontology. This is supplied by the user.

```
Retrieve service/operation names of $a$ from p-assertions
Retrieve service/operation names of preceding activity $p$ from p-assertions
$R \longleftarrow$ Get input domain-level type of $a$ from registry
$A \longleftarrow$ Get output domain-level type of $p$ from registry
Get ontology from user
```

Fig. 5. Domain-level interaction validation: computing requirement R and actual value A for activity a.

5.3 Activity validity

To support Use Cases 4 and 5, we verify that the metadata associated with services conforms to certain criteria. We use the myGrid profile [17] to identify the tasks services perform. (The myGrid profile is an extension of the OWL-S profile [7]). Likewise, the profile also specifies databases usage restrictions. Thus, the process of verifying the activity validity of an experiment involves checking that each activity's profile satisfies the requirements specified for it. The requirement can be different for each activity, as in Use Case 4. In other situations, the requirement can be the same for every activity in the workflow, such as in Use Case 5.

An activity is considered to fulfil requirement R if the metadata annotation for the operation (A) is of the same class or is a subclass of R. Figure 6 shows the algorithm used for computing the values R and A for activity a. The validator first obtains the names of the service and operation for activity a from the provenance store. It then retrieves the semantic annotations of operation a from the registry. The user supplies the requirement R for the activity. In Use Case 4, R is the original plan of the experiment. In Use Case 5, R is the set of legal requirements devised according to patenting needs. Any required ontology is also supplied by the user.

```
Retrieve service and operation names of $a$ from p-assertions
$A \longleftarrow$ Get semantic annotation of $a$ from registry
$R \longleftarrow$ Get requirements from user
Get ontology from user
```

Fig. 6. Activity validation: computing requirement R and actual value A for activity a.

After the validator computed the values R and A, it can verify whether A satisfies R, as shown in Figure 2. For Use Case 1, verification of satisfaction is performed using a validating XML parser. For the other use cases, semantic reasoning is required. This will be explained in the next section.

6 Semantic Reasoning for Validation

All of the algorithms presented in the previous section require some properties (type, legal restrictions etc.) of multiple entities to be compared. An exact match of types is inadequate for validation of an experiment, as illustrated in the examples below, and so semantic reasoning allows our architecture to take full advantage of the relationship between types encoded in ontologies. In this section, we illustrate some of the reasoning that can be employed by our validation architecture, with examples taken from a bioinformatics application in which we have tested a implementation of our architecture (see Section 7).

6.1 Validation by Generalisation

The simplest and most commonly required form of reasoning is to compare two types where one is a super-class of the other.

For example, database D may advertise its download operation as returning *RNA sequences*. Analysis service A advertises its analysis operation taking as input *nucleotide sequences*. The ontology specifies that both *DNA* and *RNA sequences* are subclasses of *Nucleotide sequences*. Therefore, the interaction between the download operation D and analysis service A is valid as the input type of A is a superclass of the output type of D.

Similarly, in Use Case 4, a plan is defined using high-level concepts to describe the operations to be performed at each stage of the experiment. For example, in the experiment plan for our sample bioinformatics application, one of the steps requires a *Compression* algorithm. The provenance records that a *PPMZ* algorithm was used in the experiment and, in the ontology, *PPMZ* algorithm is defined as a sub-class of *Compression* algorithm. Therefore, the semantic validator can verify that this operation conforms to the one in the original plan.

6.2 Validation of Inter-Parameter Constraints

The same experiment provides cases for more novel forms of semantic description and reasoning in validation. One service, *gcode*, in our bioinformatics workflow takes two parameters: a sequence and a grouping alphabet. The sequence, which may represent either an amino acid sequence or a nucleotide sequence, is encoded as a sequence of symbols. The grouping alphabet specifies a set of non-overlapping groups of symbols, each group having a symbolic name. Service *gcode* replaces each symbol in the input sequence with the name of the group to which it belongs, so that the output of the service is a sequence of group names of the same length as the original sequence.

In order for the workflow to be semantically valid, the symbols used in the input sequence of *gcode* must have the same meaning as those making up groups in the grouping alphabet. That is, if the grouping alphabet specifies groups of nucleotides (A, G, C and T/U) then the input sequence should be a nucleotide sequence, and if the alphabet specifies groups of amino acids (A, B, C, D, E...) then the input sequence should be an amino acid sequence.

The ontology contains the concepts *Sequence* and *GroupingAlphabet* both of which are parameterised on the types of their elements, which can be either *Nucleotides* and *Amino Acids*. In the registry, the *gcode* service is annotated with metadata defining the semantic types of its input parameters. We wish to advertise the fact that the arguments used as input parameters to this service must have corresponding BaseTypes: if the sequence is made up of amino acids, the alphabet should also be. That is, one is a *Sequence* with property *hasElementType* with target X, the other is is a *GroupingAlphabet* with property *hasLetterType* with target Y and X is equal to Y. Because X and Y effectively denote variables to be instantiated in different ways in different experiments, it is impossible to express this constraint with OWL alone. Instead we can use technologies such as the Semantic Web Rule Language [18] or *role-value maps* [19], with which we can express that the value of one concept's property (X) must be equal to the value of another concept's property (Y) without giving the type of those values. This mechanism has also been used to specify configuration policies of registries [20].

The input sequence and the grouping alphabet are provided to *gcode* by two other actors, and these interactions are recorded in a provenance store. From the provenance data, the element type of the input sequence and the letter type of the grouping alphabet in a particular experiment can be determined.

7 Evaluation

In this section, we present our evaluation of validation framework in satisfying two of the use cases (Use Case 2 and Use Case 4) in a sample bioinformatics experiment.

For completeness, we briefly explain the intent of the experiment used in evaluating our architecture. A more detailed description of this experiment and its workflow can be found in [21]. The experiment, developed by Klaus-Peter Zauner and Stefan Artmann, studies the structure of protein sequences by analysing their compressibility. Proteins are amino acid chains that fold into unique 3D structures. This 3D shape of the protein determines its function. The structure of protein sequences is of considerable interest for predicting which sections of the DNA encode for proteins and for predicting and designing the 3D-shape of proteins. For comparative studies of the structure present in an amino acid, it is useful to determine their compressibility. This is because compression exploits context-dependent correlations within a sequence. The fraction of its original length to which a sequence can be losslessly compressed is an indication of the structure present in the sequence.

For the evaluation, we ran the workflow multiple times and recorded the executions in the provenance store. Both the provenance store and the registry were implemented as Web Services (available for download at `pasoa.org` and `grimoires.org` respectively). The semantic validation component was implemented in Java and used Jena 2.1 for reasoning over the ontology. The ontology itself was specified in OWL and based on the ontology developed by the bioinformatics Grid project, myGrid. After a set of workflow runs, each analysing one sample, the provenance store contains records of interactions between services. Each interaction record contains the invocation message that occurred in the workflow, which specifies the operation invoked and data exchanged as arguments. In addition to the message itself, the services record data links that specify when the output of one service has been used as the input of another service. Collectively, the data links describe the data flow throughout the experiment. The full provenance data for one workflow run was approximately 1 MB in size. For the evaluation, we deployed the registry on a Windows XP PC with Pentium 4 CPU, 3 GHz, 2 GB RAM, and the provenance store and semantic validator on another Windows XP PC with Pentium 4 CPU, 1.5 GHz, 2 GB RAM. The PCs were connected by a 100Mb local ethernet. The results of each experiment is described in further detail below.

Two forms of validation were implemented, corresponding to Use Cases 2 and 4, implementing the algorithms in Figures 5 and 6 respectively. Given that we intend large numbers of experiments to be performed, it is critical to that our approach scales well as the amount of data in the provenance store expands. Therefore, we performed the validation process on all experiments in the provenance store as we increase the number of experiments. Figure 7 shows the performance of the semantic validation architecture as the number of experiments for which provenance documentation is recorded and are to be validated increases.

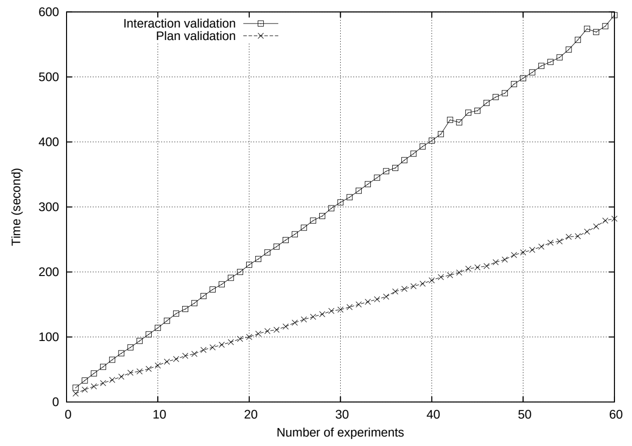

Fig. 7. Evaluation of interaction validity and conformance to plan for an increasing number of experiments.

7.1 Interaction validity, domain level

In the interaction validity experiment, the type of each output message part in the experiment was compared with the type of the input message part of the succeeding invocations in which it was used. We obtained the names of the input and output message parts from the provenance store. We then use these names to obtain their domain level types from the registry. The domain level types are ontology terms. The comparison is done by checking if the output type is the same class or a subclass of the input type. Specifically, we use the member functions `hasSuperClass` and `equals` of the `OntClass` interface in Jena. As can be seen in Figure 7, the time required for validation increases linearly with respect to the number of experiments. In total, one test included a total of $452N + 1$ Web Service calls to either the provenance store and registry plus $48N$ occurrences of reasoning using the ontology, where N is the number of experiments.

7.2 Conformance to plan

In the conformance to plan experiment, the planned data flow of each experiment is expressed using high-level concepts from the ontology to define the operations to be performed, and the service operations advertised in the registry were annotated with low-level concepts specifying the exact algorithm used by that operation. The validator checked that every data link in the provenance store corresponded to one in the plan. This is achieved by checking that the performed action is the same class or a subclass of the planned action. As can be seen in Figure 7, the time required for validation increases linearly with respect to the number of experiments. In total, one test included a total of $260N + 1$ Web Service calls to either the provenance store and registry plus $48N$ occurrences of reasoning using the ontology, where N is the number of experiments.

8 Discussion

The myGrid and CombeChem (`combechem.org`) projects have also worked on the problems of provenance recording and service description, and adopted RDF-based approaches, making ontology-based reasoning a possibility. However, neither identify the architectural elements required for validation nor provide a generic, domain-independent way to satisfy use cases such as those presented in this paper.

As our design is dictated by pragmatic considerations, we have adopted a hybrid approach to information representation. Provenance information is made available by the provenance store as XML documents. Inside the data structure representing provenance, we may find assertions, made by some actors. These assertions may be expressed using semantic web technologies. For instance, the function performed by a service or a description of its internal state expressed using OWL. In the registry, annotations provided by third parties may be encoded in the formalism of their choice. We have explicitly experimented with OWL and RDFS. Therefore, semantic reasoning (based either on OWL or RDFS technology) only operates on a subset of the provenance representation and some descriptions published in the registry.

This hybrid approach to representation suits the intensive data processing requirements of Grid applications. For roughly 10 days of computation over 100 nodes, we expect our provenance store to accumulate approximately $10 \times 100 \times 24 \times 60 \times 60 \times 20 = 1,728,000,000$ invocations to be described for the provenance of a data set. Selectively identifying the elements to reason over is therefore essential. All our use cases operate using the same reasoning pattern, which consists of identifying the provenance of some data and iterating on all its records. More advanced use cases will result in new patterns of processing. As all possible patterns cannot be anticipated, we allow users to use declarative specifications of what they expect from computations. Policy languages such as KAoS [20] are strong contenders for this problem. KAoS positive and negative obligations allow us to encode what has to occur, or what should not occur. For example, we can introduce a policy for a transactional system that requires every action to be committed or rolled back by the end of an experiment. The challenge will be to integrate the KAoS reasoner (also OWL based) with the potentially large size of the provenance store.

9 Conclusions

Grid based e-Science experiments typically involve multiple heterogeneous computing resources across a large, open and distributed network. As the complexity of experiments grows, determining whether results produced are meaningful becomes an increasingly difficult task. In this paper, we studied the problem of validation on such experiments. Traditionally, program validation is carried out either statically or at runtime. However, the usefulness of either approach is limited for large scale e-Science experiments. Static analyses rely on the availability of workflow scripts. These scripts may not be expressed in languages that analysis tools operate on, or may not be available because they are exposed as web services. Run-time service-based error checking is service dependent and users may not have control over its configuration.

We propose an alternative, provenance-based approach to experiment validation. The provenance of an experiment documents the complete process that led to the results. As a result, validation is not reliant on the availability of workflow scripts or service configurations. Moreover, as science progresses, criteria for validation evolve. Using a provenance-based approach, the validation process can be repeated without re-running the experiment. By employing technologies for provenance recording, annotation of service descriptions and semantic reasoning, we have produced an effective solution to the validation problem. Algorithms working over the automatically recorded documentation of experiments and utilising the semantic descriptions of experimental services in registries can test the validity of results to satisfy various domain-independent and domain-specific use cases.

To demonstrate the viability of our semantic validation architecture, we have discussed how it can be used with various algorithms and forms of semantic reasoning to satisfy five use cases. We have also implemented two of the use cases. Performance tests show our algorithms scale linearly as the amount of provenance documentation recorded increases.

10 Acknowledgements

This research is funded in part by the Grimoires (EPSRC Grant GR/S90843/01), myGrid (EPSRC Grant GR/R67743/01) and PASOA (EPSRC Grant GR/S67623/01) projects. The authors would also like to thank Klaus-Peter Zauner and Stefan Artmann for providing us with the bioinformatics experiment.

References

1. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. International Journal of High Performance Computing Applications **15** (2001) 200–222
2. Gil, Y., Deelman, E., Blythe, J., Kesselman, Tangmunarunkit, H.: Artificial intelligence and grids: workflow planning and beyond. IEEE Intelligent Systems **19** (2004) 26–33
3. Consortium, T.G.O.: The Gene Ontology (GO) database and informatics resource. Nucleic Acids Research **32** (2004) 258–261
4. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web services description language (WSDL) 1.1. Technical report, W3C Note, `http://www.w3.org/TR/wsdl` (2001)
5. Fallside, D.C., Walmsley, P.: XML schema part 0: Primer second edition. Technical report, W3C Recommendation, `http://www.w3.org/TR/xmlschema-0` (2004)

6. Mitra, N.: SOAP version 1.2 part 0: Primer. Technical report, W3C Recommendation, `http://www.w3.org/TR/soap12-part0` (2004)
7. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: OWL-S: Semantic markup for web services. Technical report, W3C Member Submission, `http://www.w3.org/Submission/OWL-S` (2004)
8. Wroe, C., Goble, C., Greenwood, M., Lord, P., Miles, S., Papay, J., Payne, T., Moreau, L.: Automating experiments using semantic data on a bioinformatics grid. IEEE Intelligent Systems **19** (2004) 48–55
9. Miles, S., Papay, J., Luck, M., Moreau, L.: Towards a protocol for the attachment of metadata to grid service descriptions and its use in semantic discovery. Scientific Programming **12** (2004) 201–211
10. Lee, M., Han, D., Shim, J.: Set-based access conflicts analysis of concurrent workflow definition. In: Proceedings of Third International Symposium on Cooperative Database Systems and Applications, Beijing, China (2001) 189–196
11. Baresi, L., Maurino, A., Modafferi, S.: Workflow partitioning in mobile information systems. In: Proceedings of IFIP TC8 Working Conference on Mobile Information Systems (MOBIS 2004), Oslo, Norway, Springer (2004) 93–106
12. Eshuis, R., Wieringa, R.: Verification support for workflow design with uml activity graphs. In: Proceedings of the 24th International Conference on Software Engineering. (2002) 166–176
13. Yang, L., Bundy, A., Berry, D., Huczynska, S.: Inferring quality of service properties for grid applications. In: CS poster, EPSRC e-Science Meeting, Edinburgh, UK, NeSC (2004) static analysis of workflows.
14. Blythe, J., Deelman, E., Gil, Y.: Planning for workflow construction and maintenance on the grid. In: ICAPS 2003 workshop on planning for web services. (2003)
15. Uszok, A., Bradshaw, J.M., Jeffers, R.: KAOS: A policy and domain services framework for grid computing and semantic web services. In Jensen, C., Poslad, S., Dimitrakos, T., eds.: Trust Management: Second International Conference (iTrust 2004) Proceedings. Volume 2995 of Lecture Notes in Computer Science., Oxford, UK, Springer (2004) 16–26
16. Miles, S., Groth, P., Branco, M., , Moreau, L.: The requirements of recording and using provenance in e-science experiments. Technical report, Electronics and Computer Science, University of Southampton (2005)
17. Wroe, C., Stevens, R., Goble, C., Roberts, A., Greenwood, M.: A suite of DAML+OIL ontologies to describe bioinformatics web services and data. nternational Journal of Cooperative Information Systems **12** (2003) 197–224
18. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A semantic web rule language combining OWL and RULEML. Technical report, DARPA Agent Markup Language (DAML) Program, `http://www.daml.org/2003/11/swrl/` (2003)
19. Schmidt-Schauss, M.: Subsumption in KL-ONE is undecidable. In Brachman, R.J., Levesque, H.J., Reiter, R., eds.: Proceedings of the 1st International Conference on the Principles of Knowledge Representation and Reasoning (KR89), Morgan Kaufmann (1989) 421–431
20. Moreau, L., Bradshaw, J., Breedy, M., Bunch, L., Johnson, M., Kulkarni, S., Lott, J., Suri, N., Uszok, A.: Behavioural specification of grid services with the KAOS policy language. In: Proceedings of Cluster Computing and Grid (CCGrid), Cardiff, UK (2005)
21. Groth, P., Miles, S., Fang, W., Wong, S.C., Zauner, K.P., Moreau, L.: Recording and using provenance in a protein compressibility experiment. In: Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14), NC, USA (2005)