March 2007
Volume 5
ISSN 1570-8268
ELSEVIER
JOURNAL OF
Web Semantics
SCIENCE, SERVICES & AGENTS ON THE WORLD WIDE WEB

Available online at www.sciencedirect.com
ScienceDirect
Web Semantics: Science, Services and Agents on the World Wide Web 5 (2007) 28–38
JOURNAL OF Web Semantics
www.elsevier.com/locate/websem

Provenance-based validation of e-science experiments

Simon Miles [*], Sylvia C. Wong, Weijian Fang, Paul Groth, Klaus-Peter Zauner, Luc Moreau

School of Electronics and Computer Science, University of Southampton, Highfield, Southampton SO17 1BJ, United Kingdom

Received 15 June 2006; received in revised form 30 October 2006; accepted 30 November 2006
Available online 4 January 2007

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, not necessarily anticipated prior to execution. Scientists may also want to review and verify experiments performed by their colleagues. There are no existing 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.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Provenance; E-science; Process validation; Semantic service description

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 [8].

E-science experiments are typically formed by invoking multiple services, whose compositions are modelled as workflows [9]. 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 or a workflow before it is executed and establishes that the program/workflow 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 when workflows are executed in dynamic 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

[*] Corresponding author. Tel.: +44 23 8059 8309.
E-mail addresses: sm@ecs.soton.ac.uk (S. Miles), sw2@ecs.soton.ac.uk (S.C. Wong), wf@ecs.soton.ac.uk (W. Fang), pg03r@ecs.soton.ac.uk (P. Groth), kpz@ecs.soton.ac.uk (K.-P. Zauner), l.moreau@ecs.soton.ac.uk (L. Moreau).

1570-8268/$ – see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.websem.2006.11.003

result, no guarantee exists that specific, domain-level types will be checked dynamically.

A new, more specific limitation comes from the evolving conduct of e-science. 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, models 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. We have studied a range of e-science application domains and established that they have a range of requirements for provenance-awareness [15]. In the former requirements study, many examples of experiment validation were discovered and in varying domains. For example, in a distributed particle physics experiment, there was a requirement to verify that those library versions used to analyse the experiment data were not ones known to contain bugs. In partially lab-based biology and chemistry experiments, requirements for validation included checking that health and safety rules had been followed by experiments in the past month. In a more general, computer-science centred example [25], processes can be validated to ensure, for fault tolerance, that multiple services assumed to be independent did not actually depend on the same, possibly faulty, service. We refer the reader to the survey for the full range of such use cases.

In this paper, our thesis is that provenance-based validation of experiments allows us to verify their 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. As identified in the use cases below, while service-based validation can only be performed at run-time, it is sometimes necessary to validate an experiment after it has been executed. 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, because criteria evolve as science progresses. Thus, it is important that previously computed results can be verified according to revised sets of criteria.

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 that DNA and RNA sequences are subtypes of nucleotide sequences, as illustrated by the Gene Ontology [24]. 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). Therefore, to ensure results of the experiment are not affected by this error in the ontology, *B* validates the execution against the revised ontology. For instance, if at the time that the experiment was performed, the ontology erroneously included an assertion that the class NucleotideSequence subsumed the class AminoAcidSequence, then a workflow using a service expecting nucleotide sequences would be determined valid when applied to amino acid sequences. After correction, the workflow can be seen to be no longer valid. The value, in this case, of validation after experiments have taken place is that information can be assumed to be as accurate as possible before workflow execution.

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 they designed and performed the experiment. In this case, it may not be a lack of information available before workflow execution, but a lack of requirement to validate that means validation will not occur. As another example, new scientific results published in the literature may introduce a doubt about the quality of part of a database. The scientist may wish to ensure that the data they used in a past experiment, did not come or derive from this database. In general, our approach helps the case where, for one reason or another, the biologist chooses not to validate or does not even consider validating the experiment before execution as, through our approach, the facility is readily available after execution.

This list of use cases is by no means exhaustive. We have focussed on forms of validation requiring minimal additional input from the validating user, and so do not address those concerning the scientific intent of the experiment. Other use cases are made explicit elsewhere [15], some of which are examined in other papers [10], while others remain for future work.

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 [6]. In other words, it is the type expected by the transport layer (i.e. SOAP [17]).

In our approach, we allow anyone to 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 [14] 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 coexist. Hence, it is permitted to associate several semantic types to a given entity: this is the approach adopted by myGrid [28], 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 Grimoires registry [16].

In Section 1, 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 [13], graph-based partitioning of workflows [1], model checking of activity graphs [5] and checking the protocols of Web Service composition by multi-agent systems [27].

Validation may also 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

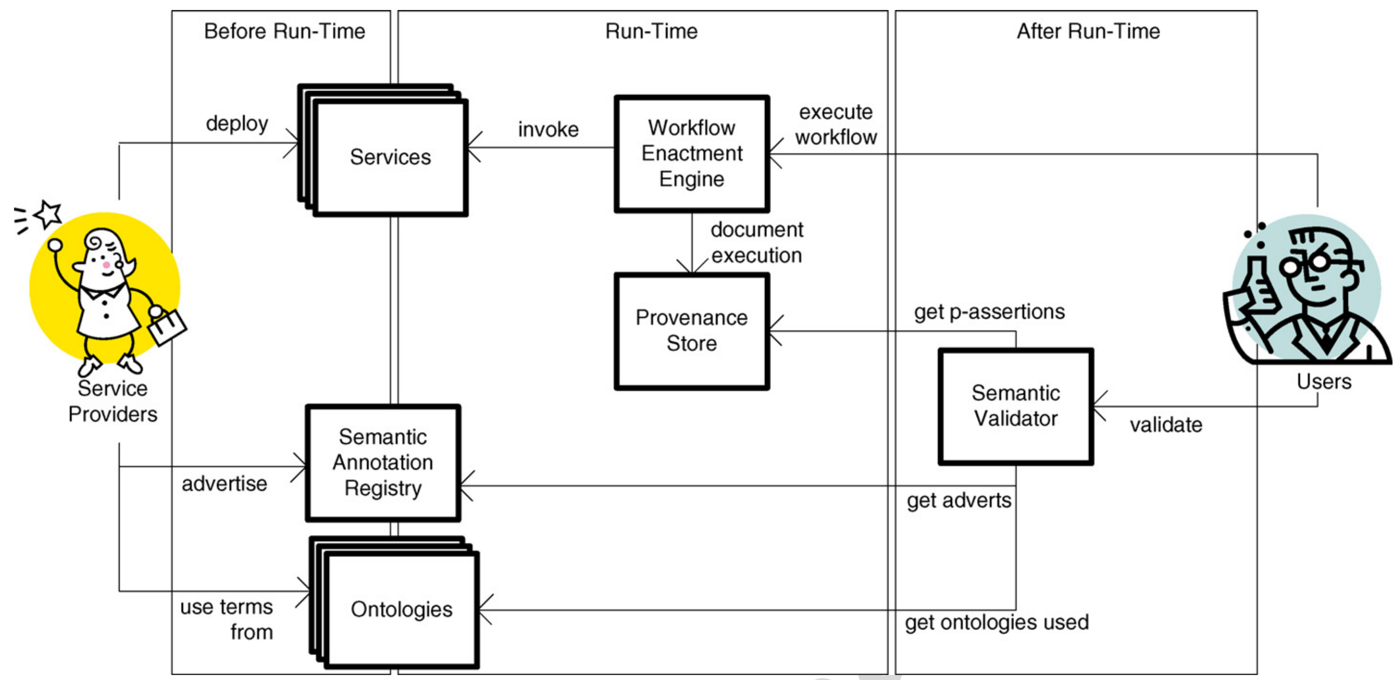

Fig. 1. Provenance-based validation architecture.

in a workflow employ validating parsers, the workflow execution is guaranteed to satisfy syntactic types required by services. We note however that many of the XML deserialisers incorporated into services, such as Apache Axis (ws.apache.org/axis) and JAXB (java.sun.com/xml/jaxb), do not perform validation by default because validation is an expensive operation that affects the performance of the 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 [2]. 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 [26].

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 data regarding provenance with other scientists. Also, as validation criteria evolve, the validation process can be repeated without re-executing the experiment.

Fig. 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 [16]. 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. The annotations make use of one or more ontologies, as preferred by each service provider.

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. Each message sender and receiver (collectively, *actors*) in an experiment can assert facts about that experiment, called *p-assertions* (assertions, by an actor, pertaining to a process). A p-assertion may state one of three facts. First, the content of a message sent by one actor to another can be asserted as an *interaction p-assertion*. Second, the causal relationship between the inputs and outputs of an actor due to its processing can be asserted as a *relationship p-assertion*. Finally, the state of an actor when an interaction took place can be asserted as 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. Collectively, the p-assertion contents of a provenance store is called *process documentation*.

After the experiment is carried out, validation is performed using the algorithm outlined in Fig. 2. Validation is done for each

```
is Valid ← true
for all activities a ∧ is Valid do
    (R, A) ← compute(a)
    is Valid ← A satisfies R
endfor
```

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

```
Function: Compute requirement R and actual value A
Require: Activity a
  Retrieve p-assertions from provenance store for activity a
  Get advertisements from registry for activity a
  Get user supplied information for activity a
  R ← Compute required value for a from above data
  A ← Compute actual value for a from above data
```

Fig. 3. Algorithm to compute required and actual values *R* and *A*.

service operation in the experiment. When retrieving service descriptions, the ontologies their annotations refer to are also retrieved so that reasoning can be performed. 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.

Fig. 3 explains how requirement *R* and actual value *A* are calculated for a given activity *a*. First, the validator obtains p-assertions for *a* from the provenance store. Using this 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.

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

Fig. 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 Fig. 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 Fig. 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 p-assertions
R ← Get message part type in WSDL document for a from advertisements
A ← Retrieve input to a from p-assertions
```

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*. Fig. 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 referred to by the ontology terms themselves as used in the registry annotations.

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 [29] to identify the tasks services perform. (The myGrid profile is an extension of the OWL-S profile [14].) 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.

```
Retrieve service/operation names of a from p-assertions
Retrieve service/operation names of preceding activity p from p-assertions
R ← Get each input domain-level type of a from advertisement
A ← Get output domain-level type of SAME ARGUMENT OF p from advertisement
```

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

```
Retrieve service and operation names of a from p-assertions
Get ontology from user
A ← Get semantic type of a from advertisements
R ← Get required activity type of a from user
```

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

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*. Fig. 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.

After the validator computed the values *R* and *A*, it can verify whether *A* satisfies *R*, as shown in Fig. 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.

There is nothing in our architecture to prevent conflicts in the assertions of the ontologies used by the service providers in advertising services, and the user in describing the planned activities for validation. Similarly, there is nothing here to prevent multiple service providers from using conflicting ontologies in advertising their services, affecting the value of the interaction validity. Detecting and handling such conflicts is a wider issue for the Semantic Web and not within scope of our work, but we note that by validating workflows after execution, it is possible to take advantage of the removal of ontology conflicts over time to perform increasingly valuable validation. Additionally, because our registry allows third parties to add annotations to service descriptions, it is perfectly possible that the user of a service and the publisher of annotations to that service are the same person. In the evaluation presented in the next section, we have used a single ontology without conflicts.

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 super-class 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 process documentation records that a *PPMZ* algorithm was used in the experiment and, in the ontology, *PPMZ* algorithm is defined as a subclass 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, *Encode by Groups*, in our bioinformatics workflow (full description in the next section) 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 [11] or *role-value maps* [22], 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 [18].

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 process documentation, 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 the validation framework in satisfying two of the use cases (Use Cases 2 and 4) in a sample bioinformatics experiment.

7.1. Experiment

The experiment used in evaluating our architecture was designed by Klaus-Peter Zauner and Stefan Artmann.

7.1.1. Biology

Proteins are the essential functional components of all known forms of life; they are linear chains of typically a few hundred building blocks taken from the same set of about 20 different amino acids. Protein sequences are assembled following a code sequence represented by another polymer (mature mRNA). This polymer is produced by splicing certain pieces (the exons) of a molecular copy of the coding region of a gene on the DNA, while discarding other pieces (the introns) of the copy. During and following the assembly, the protein will curl up under the electrostatic interaction of its thousands of atoms into a defined but agile shape of typically 58 nm size. The resulting 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. Compression exploits context-dependent correlations within the sequence. The fraction of its original length to which a sequence can be loss-lessly compressed is an indication of the structure present in the sequence. In general, no practical compression method can discover all structures, so actual compression of a sequence can only yield a lower bound of the sequences compressibility. For the same reason, the compressibility values are also relative to the applied compression method [12]. Methods that are good in discovering structure are computationally expensive; initial investigations on protein compressibility indicated that it is indeed difficult to discover structure in protein sequences [20]; however, recently, progress has been made by grouping amino acids [21]: if the compression of the sequences serves only to quantify structure and decompression is not intended, the sequences can be recoded with a reduced alphabet. In an amino acid sequence, for instance, each amino acid symbol is replaced by a symbol representing a group of amino acids. Compression is then applied to the recoded sequence. The results of this experiment can, for example, be used to determine the amino acid groupings that maximise compressibility.

7.1.2. Workflow

The main workflow of the comparative sequence compressibility experiment is shown in Fig. 7. Some stages of the workflow (named Measure in the figure) are sub-workflows depicted in Fig. 8. It starts with the selection of a sequence sample, which sample may be composed from several individual sequences to provide enough data for the statistical methods employed by the compression algorithms (Collate Sample). This sample is then recoded with a given group coding (Encode by Groups). The recoded sequence is then compressed with compression algorithms, e.g. gzip, bzip2 or ppmz, to obtain the length of the compressed sequence (as seen in Fig. 8). Random permutations of the sequence (Shuffle) are also compressed to provide a standard for comparison. This standard removes the influence of two factors from the calculation of compressibility: the particular data encoding used to represent the groups, and the non-uniform frequency of groups. From the results, a compressibility value is obtained for the sample sequence that is relative to both the compression method and group coding employed. The variability in the compressed length of the permuted sequences leads to a distribution of compressibility values (Collate Sizes). The workflow entails a sufficient number of compressions of permuted sequences to estimate the standard deviation for the compressibility (Average).

7.2. Test procedure

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 relationships that specify when the output of one service has been used as the input of another service. Collectively, the relationships describe the data flow throughout the experiment. The full process documentation for one workflow run consisted of 120 p-assertions: 60 interaction p-assertions recording the messages sent between services, 30 relationship p-assertions asserting the functions applied to that message data within services and 30 actor state p-assertions recording the contents of the scripts which processed the data. This equated to approximately 1 MB on disk for one experiment's process documentation. 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 100 Mb local ethernet. The results of each experiment is described in further detail below.

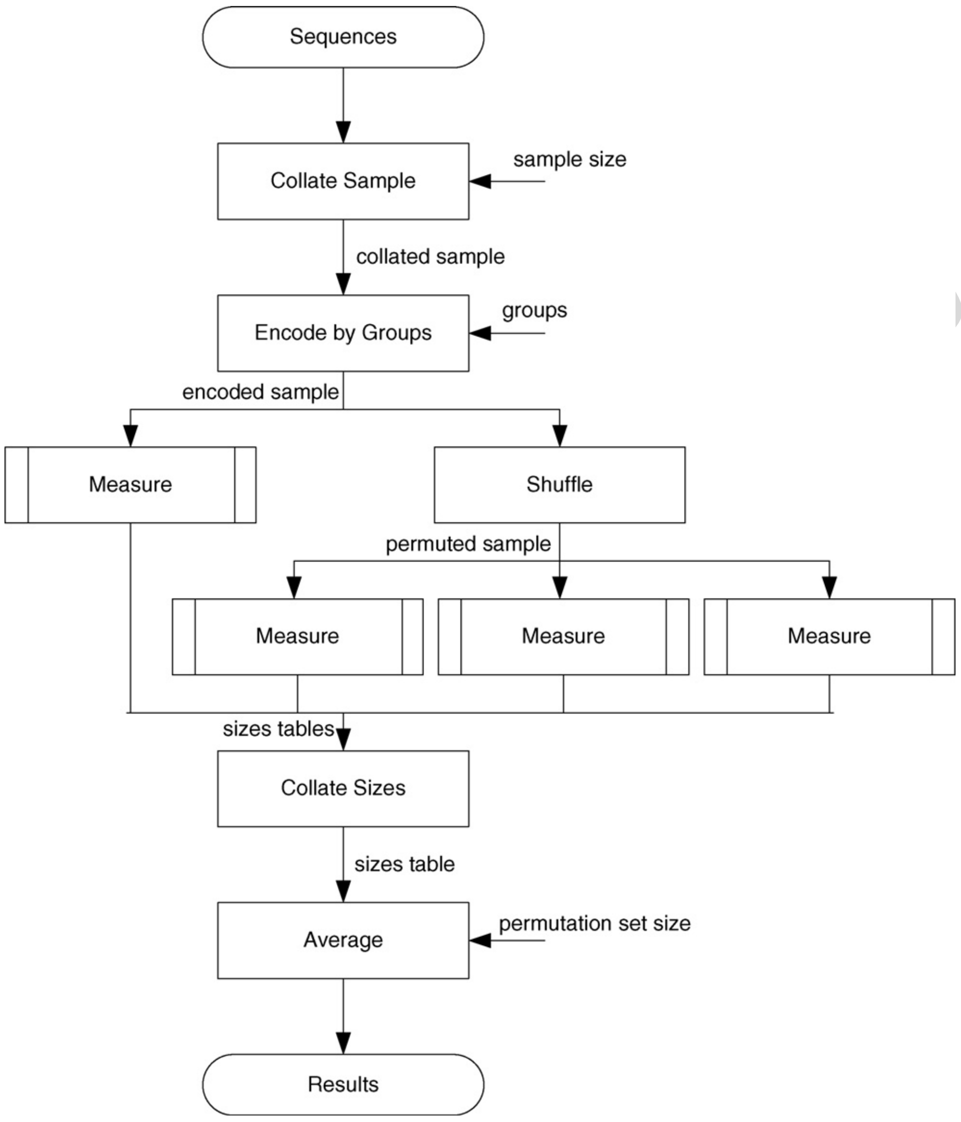

Fig. 7. Main protein compressibility experiment workflow.

Two forms of validation were implemented, corresponding to Use Cases 2 and 4, implementing the algorithms in Figs. 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. Fig. 9 shows the performance of the semantic validation architecture as the number of experiments for which process documentation is recorded and are to be validated increases.

7.3. 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 Fig. 9, the time required for validation increases linearly with respect to the number of experiments. Overall, 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.

7.4. 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

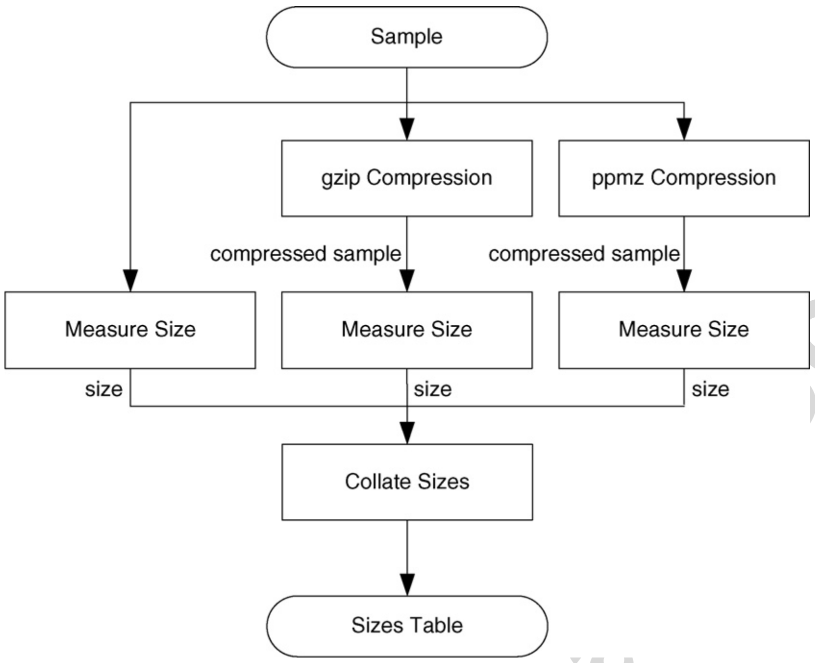

Fig. 8. Measure sample sub-workflow.

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 Fig. 9, 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.

7.5. User feedback

We returned to the scientist to ask about the benefits of the approach described in this paper. Without the provenance-based validation system, determining the answer to the two use cases evaluated required inferring the types of data from their file-names and trawling through log files to identify script (service) names and versions.

The primary advantages noticeable from using a provenance-based approach are the well-defined structure of the data recorded and the connections between data from disparate parts of the experiment: in practice, this allows the exact information required to validate or determine differences between experiments to be extracted and processed or presented. The scientist also highlighted the benefit of having remote, independent stores for process documentation as compared to using local log files, which became too large to manageably traverse even without the detail recorded in our provenance-based system.

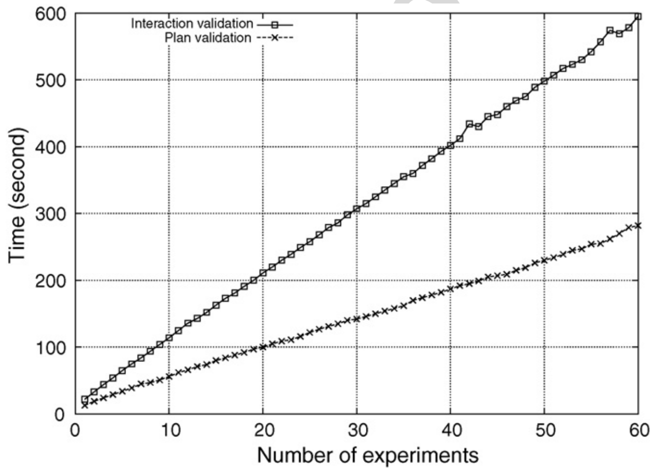

Fig. 9. Evaluation of interaction validity and conformance to plan for an increasing number of experiments.

8. Discussion

There are other approaches to recording and using provenance in existence. For example, the myGrid and CombeChem (combechem.org) projects have also worked on the problems of recording data for determining provenance, and of service description, and adopted RDF-based approaches, making ontology-based reasoning a possibility. Comprehensive surveys of recent provenance-related work already exist [3,23], so we do not expand on them further here, but note that they do not 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. Process documentation is made available by the provenance store as XML documents following open specifications [19]. Inside the data structure, 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. More recent performance statistics for the registry have been published separately [7].

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 documentation regarding approximately $10 \times 100 \times 24 \times 60 \times 60 \times 20 = 1{,}728{,}000{,}000$ invocations. 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 [18] 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 run-time. 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 process documentation recorded increases.

Acknowledgements

This research is funded in part by the Grimoires (EPSRC Grant GR/S90843/01) and PASOA (EPSRC Grant GR/S67623/01) projects.

References

[1] L. Baresi, A. Maurino, S. Modafferi, Workflow partitioning in mobile information systems, in: Proceedings of IFIP TC8 Working Conference on Mobile Information Systems (MOBIS 2004), Springer, Oslo, Norway, 2004, pp. 93–106.
[2] J. Blythe, E. Deelman, Y. Gil, Planning for workflow construction and maintenance on the grid, in: Proceedings of ICAPS 2003 Workshop on Planning for Web Services, 2003, pp. 8–14.
[3] R. Bose, J. Frew, Lineage retrieval for scientific data processing: a survey, ACM Comput. Surv. 37 (March (1)) (2005) 1–28.
[4] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/wsdl, March 2001.
[5] R. Eshuis, R. Weiringa, Verification support for workflow design with UML activity diagrams, in: Proceedings of the 24th International Conference on Software Engineering, 2002, pp. 166–176.
[6] D.C. Fallside, P. Walmsley, XML Schema part 0: Primer, second ed., http://www.w3c.org/TR/xmlschema-0, 2004.
[7] W. Fang, S.C. Wong, V. Tan, S. Miles, L. Moreau, Performance analysis of a semantics enabled service registry, in: S.J. Cox, D.W. Walker (Eds.), Proceedings of the UK e-Science All Hands Meeting 2005, Nottingham, UK, September, EPSRC, 2005 (proceedings published on CD).
[8] I. Foster, C. Kesselman, S. Tuecke, The anatomy of the grid: enabling scalable virtual organizations, Int. J. Supercomput. Appl. 15 (3) (2001) 200–222.
[9] Y. Gil, E. Deelman, J. Blythe, C. Kesselman, H. Tangmunarunkit, Artificial intelligence and grids: workflow planning and beyond, IEEE Intell. Syst. 19 (2004) 26–33.
[10] P. Groth, S. Miles, W. Fang, S.C. Wong, K.-P. Zauner, L. Moreau, Recording and using provenance in a protein compressibility experiment, in: Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC'05), IEEE Computer Society, July 2005, pp. 201–208.
[11] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, SWRL: A Semantic Web Rule Language Combining OWL and RuleML, http://www.daml.org/2003/11/swrl/, November 2003.
[12] K. Lanctot, M. Li, E.H. Yang, Estimating DNA sequence entropy, in: Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, California, January 9–11, ACM, 2000, pp. 409–418.
[13] M. Lee, D. Han, J. Shim, Set-based access conflicts analysis of concurrent workflow definition, in: Proceedings of the Third International Symposium on Cooperative Database Systems and Applications, Beijing, China, 2001, pp. 189–196.
[14] D. Martin, M. Burnstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N.

Srinivasan, K. Sycara, OWL-S: Semantic Markup for Web Services, http://www.daml.org/services/owl-s/1.0/, 2006.
[15] S. Miles, P. Groth, M. Branco, L. Moreau, The requirements of using provenance in e-science experiments, J. Grid Comput., 25 pp., doi:10.1007/s10723-006-9055-3, in press.
[16] S. Miles, J. Papay, T. Payne, M. Luck, L. Moreau, Towards a protocol for the attachment of metadata to service descriptions and its use in semantic discovery, Sci. Program. 12 (4) (2004) 201–211 (extended version of AxGrids 2004 paper for special journal issue).
[17] N. Mitra, SOAP Version 1.2 part 0: Primer, http://www.w3.org/TR/soap12-part0, 2004.
[18] L. Moreau, J. Bradshaw, M. Breedy, L. Bunch, M. Johnson, S. Kulkarni, J. Lott, N. Suri, A. Uszok, Behavioural specification of grid services with the KAoS policy language, in: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'05), Cardiff, UK, May, 2005.
[19] S. Munroe, P. Groth, S. Jiang, S. Miles, V. Tan, L. Moreau, J. Ibbotson, A. Biller, J. Vazquez-Salceda, The Open Provenance Specification, http://twiki.gridprovenance.org/bin/view/Provenance/OpenSpecification, October 2006.
[20] C. Nevill-Manning, I. Witten, Protein is incompressible, in: J. Storer, M. Cohn (Eds.), Proc. Data Compression Conference, IEEE Press, Los Alamitos, CA, 1999, pp. 257–266.
[21] G. Sampath, A block, coding method that leads to significantly lower entropy values for the proteins and coding sections of haemophilus influenzae, in: Proceedings of the Computational Systems Bioinformatics (CSB'03), IEEE Computer Society, 2003.
[22] M. Schmidt-Schauss, Subsumption in KL-ONE is undecidable, in: R.J. Brachman, H.J. Levesque, R. Reiter (Eds.), Proceedings of the First Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'89), Morgan Kaufmann, Los Altos, 1989, pp. 421–431.
[23] Y. Simmhan, B. Plale, D. Gannon, A survey of data provenance in e-science, SIGMOD Rec. 34 (3) (2005) 31–36.
[24] The Gene Ontology Consortium, The gene ontology (GO) database and informatics resource, Nucl. Acids Res. 32 (2004) 258–261.
[25] P. Townend, P. Groth, J. Xu, A provenance-aware weighted fault tolerance scheme for service-based applications, in: Proceedings of the Eighth IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 2005), May 2005, pp. 258–266.
[26] A. Uszok, J.M. Bradshaw, R. Jeffers, KAoS: a policy and domain services framework for grid computing and semantic web services, in: C. Jensen, S. Poslad, T. Dimitrakos (Eds.), Trust Management: Second International Conference (iTrust 2004) Proceedings, vol. 2995, Oxford, UK, March, Springer, 2004, pp. 16–26.
[27] C.D. Walton, Model checking multi-agent web services, in: Proceedings of the 2004 Spring Symposium on Semantic Web Services, Stanford, California, US, 2004.
[28] C. Wroe, C. Goble, M. Greenwood, P. Lord, S. Miles, J. Papay, T. Payne, L. Moreau, Automating experiments using semantic data on a bioinformatics grid, IEEE Intell. Syst. 19 (1) (2004) 48–55.
[29] C. Wroe, R. Stevens, C. Goble, A. Roberts, M. Greenwood, A suite of DAML + OIL ontologies to describe bioinformatics web services and data, Int. J. Cooperative Inf. Syst. 12 (2003) 197–224.