

An Identity Crisis in The Life Sciences

Jun Zhao, Carole Goble, and Robert Stevens

School of Computer Science, University of Manchester, M13 9PL, U.K.
zhaoj,carole,robert.stevens@cs.man.ac.uk

Abstract. ^{my}Grid is an e-Science project assisting life scientists to build workflows that gather and co-ordinate data from distributed, autonomous, replicated and heterogeneous resources. The provenance logs of workflow executions are recorded as RDF graphs. The log of one workflow run is used to trace the history of its execution process; however, by aggregating provenance logs of workflow reruns, or runs of different workflows, we can gather the provenance of a common data product shared in multiple derivation paths. This aggregation relies on accurate and universal identification of each data product. The nature of bioinformatics data and services, however, makes this difficult. We describe the identity problem in bioinformatics data, and present a protocol for managing identity co-references and allocating identity to collected and computed data products. The ability to overcome this problem means that the provenance of workflows in bioinformatics and other domains can be themselves exploited to enhance the practice of e-Science.

1 Introduction

^{my}Grid (<http://www.mygrid.org.uk>) is an e-Science project providing middleware services to assist bioinformaticians to perform *in silico* experiments [1]. We use workflows to orchestrate, access and interoperate a large number of public databases and applications, and manage those experiments and their outcomes (including data products, their provenance and experiment conclusions) using semantic-based metadata and data technologies [2]. Our workflow environment and workbench, Taverna, enables scientists to design and execute workflows, providing access to over 3,000 bio-resources. These services are a mixture of Web services, Grid services, Java applications, database queries and scripts. Taverna has been used for gene alerting, gene and protein sequence annotation, proteomics, functional genomics, chemoinformatics, systems biology and protein structure prediction applications. Workflows have been used to identify a mutation associated with the autoimmune disorder Graves' Disease in the I kappa B-epsilon gene [3] and build the first complete and accurate map of the region of chromosome 7 involved in Williams-Beuren Syndrome (WBS) [4].

Abstract workflow designs, written in Taverna's Scuff language [4], are instantiated by specific data, parameter settings and the end points of the services to be executed to become a workflow run. Collections of workflow designs and workflow runs contribute to an overall experiment. ^{my}Grid collects both data produced during workflow runs and the provenance of these data products [2]. By

convention, the provenance log of one workflow run is used to trace: the history of the execution process (e.g. services used); the origin of a data product (e.g. the database or intermediate data product); the ownership and intellectual property of each run (e.g. who, when). Ownership, purpose, etc are provenance annotations over experimental collections of workflow designs as well as an individual workflow run. The provenance of each data product, collected or computed, is captured by detecting events during workflow enactment to form a dependency graph. Annotations of ownership and purpose are assertions of fact or opinion on the data, processes or sub-graphs.

The Taverna workflows developed for the Life Sciences have a number of properties that make an impact on our provenance collection:

- the workflows are largely data pipelines, frequently generating data collections and then iterating in turn over each item in the collection. Collections containing a bag of atomic or collection data are produced by iterations.
- a workflow data product can be (a) a newly generated original data object, or (b) a pre-existing data object gathered from an external resource. Thus, when we refer to data products, these can be pre-existing objects that have been retrieved from an external collection (*collected data*) or freshly computed data values (*computed data*).
- as the public resources regularly change their content, the same workflow is rerun repeatedly over the same resources, perhaps with different parameter settings, but often not so. The data products acquired by different runs are compared, merged and aggregated. A workflow can thus be executed repeatedly by the same user at a different time or location, or by different users from different research groups or institutions.
- the same data product may be acquired by very different workflows under different experimental contexts.

The final two points are important. Bioinformatics is an exploratory scientific discipline. Varying the settings of repeated executions might lead to completely different outcomes. Results from repeated executions need to be accumulated to verify an ultimate conclusion. Thus we need to put our provenance records to use: to aggregate, integrate and compare the provenance records gathered from multiple derivation paths for a common, shared data product collected or computed by multiple workflow runs. Consider data product d_i , acquired in both workflow runs r_1 and r_2 :

- provenance *aggregation* for d_i is the union of the provenance records gathered cross-runs, i.e. r_1 and r_2 .
- provenance *integration* for d_i is the merge of the aggregated provenance records.
- provenance pair-wise *comparison* for d_i for two runs r_1 , r_2 is the computed difference between the two provenance records.

For a scientist, the convergence of provenance can highlight its importance within the results.

To aggregate and integrate provenance metadata for the same data product arising from different workflow runs we need two things:

some mechanism for merging provenance graphs. In ^{my}Grid we have represented our workflow provenance metadata using technologies drawn from the semantic web community, chiefly the Resource Description Framework (RDF) (<http://www.w3.org/RDF/>). RDF is essentially a simple graph data model. RDF data store and query languages provide mechanisms for graph fusion and graph manipulation, as well as queries.

some mechanism for managing data product identity. When we aggregate provenance, it is hugely helpful if the same data object - a protein sequence, a gene, a database entry - has the same identity regardless of its origin. As our workflows retrieve database entries from public databases and add them to collections, we would like to use the same identifier every time we handle the same collected data object. RDF provides an explicit identification system (Universal Resource Identifiers (URIs)) for resources to allow metadata about a resource to be merged from several sources. We identify externally stored data objects, computed data products, workflows, parameters, etc by an LSID (Life Science Identifier) [5], which is a URI, an example of a global unique identifier (GUID). In many e-Science domains, such as chemistry, physics, astronomy, etc, GUIDs for data objects are taken for granted. The Digital Object Identifier (DOI) [6] provides persistent GUIDs for digital publications. LSIDs are proposed as GUIDs by the life science community. This scheme has been adopted by major life science databases, such as NCBI, UniProt, and Affymetrix ¹.

Representing provenance using RDF and LSIDs enables us to potentially aggregate multiple provenance records for a data product from individual experiment runs. Although the RDF-based graph model and associated manipulation and query mechanisms lend themselves to the support of cross-run or cross-workflow provenance aggregation and integration, the allocation and management of identity is problematic. Our identity allocation scheme works well when working with the provenance of a single run, but cross-run provenance aggregation and integration raises issues.

The rest of the paper is organized as follows: Section 2 describes the motivation for managing data identities by describing the provenance record for a simple Taverna workflow. Section 3 highlights the external identity landscape, showing how the difficulties in allocating identities in a coherent fashion has consequences on the recording and aggregating of provenance records. We analyze our problems with duplicated identities when handling externally identified pre-existing data whilst internally allocating identity to computed data arising from our workflow executions. In Section 4, we propose a new identity protocol to construct identity co-references and the introduction of a new identity naming scheme. We show how the identity co-references can help with comparing provenance generated in repeated runs. Related identity work and identity management in provenance are described in Section 5. We conclude with a discussion and summary of the characteristics of the identity problem.

¹ <http://lsid.biopathways.org/authorities.shtml>

2 Collecting provenance from a Taverna workflow

One of the most commonly used applications in bioinformatics experiments is BLAST (Basic Local Alignment Search Tool) sequence alignment application [7], which provides a method for rapid searching of nucleotide and protein databases. BLAST detects local as well as global alignments, detecting regions of similarity embedded in otherwise unrelated proteins or nucleic acids.

Figure 1 schematically presents a simple workflow (WF1), the data products it consumes and produces and the identities of those data products. The workflow is a data pipeline workflow from the WBS study. It identifies a collection of protein sequences, drawn from a database, similar to the query sequence (Seq_0). There are three steps. Step 1 is a call to a BLAST service, which takes as input a sequence (Seq_0) to align against, and a set of parameter settings (P_0) which includes the **database** ($database_0$) of sequences, the statistical significance threshold **e-value** ($evaluate_0$) for reporting sequence matches, and the maximum number of reported **scores** ($score_0$) in the output BLAST report.

A BLAST report file, $Brpt_j$ (where $j = 0..n$ representing the number of the experiment run), emerges as the output in run r_j , containing a collection of sequence data entries Seq_i s retrieved from the sequence database: $Brpt_j = \{Seq_1, Seq_2...Seq_n\}$. The sequences and their identifiers are embedded as text in the report, along with other materials. At step 2 in Figure 1, **BLAST_Simplifier** parses $Brpt_0$, and extracts the sequences to recover a collection of Seq data objects: $SEQ_j = \{i|Seq_i, 0 \leq i \leq n\}$ (where $j = 0..n$).

Finally, in step 3 the **GenBank_retrieve** service iterates in turn over each Seq in the data collection SEQ_j produced in one run of WF1, queries the GenBank database for each Seq and produces a collection ($GBRPT_j$) (where $j = 0..n$) with a text report ($GBrpt_i$) for each Seq : $GBRPT_j = \{i|GBrpt_i, 0 \leq i \leq n\}$.

Provenance cross-runs need to be collected when WF1 is re-run in three different ways:

- Case 1** using the same BLAST service and same parameter settings to obtain updated data products. As the public databases are frequently updated, WF1 in Figure 1 needs to be frequently repeated in order to collect updated sequences.
- Case 2** using the same BLAST service but different parameter settings to look for the best parameter settings for this workflow, since different parameter settings might lead to different BLAST reports.
- Case 3** using different BLAST services to look for the best service for this workflow: several BLAST services have been implemented, including one by DDBJ (DNA Databank of Japan)², one within Soaplab³ developed by the European Bioinformatics Institute (EBI) as part of the ^{my}Grid project and another by the Washington University hosted by the EBI (WU-BLAST)⁴. Different BLAST services are based on different algorithm versions or query

² www.xml.nig.ac.jp/wsd1/index.jsp

³ <http://industry.ebi.ac.uk/soaplab>

⁴ <http://blast.wustl.edu/>

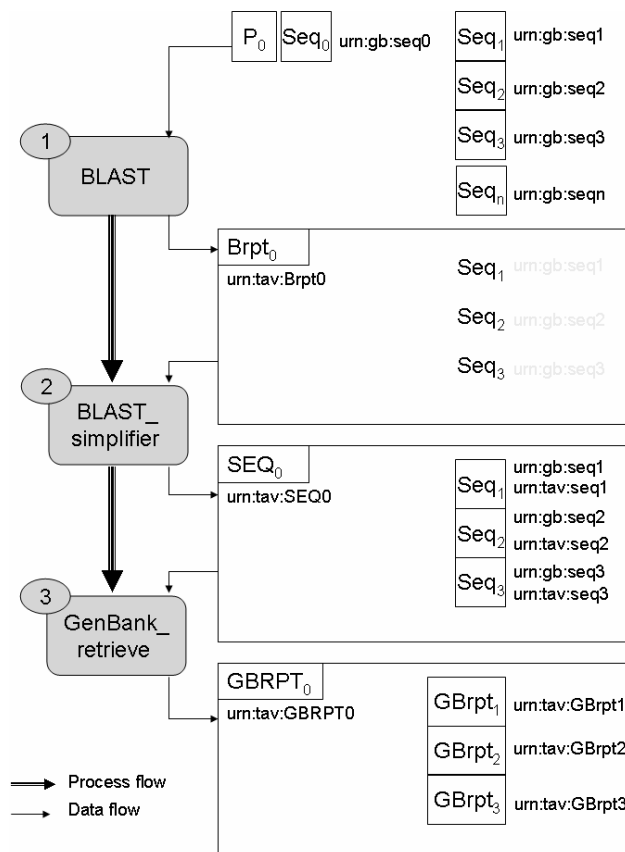


Fig. 1. Workflow (WF1) which forms part of the WBS case study. $Brpt_0$, SEQ_0 , $GBRPT_0$ and $GBrpt_i$ are computed data products. Seq_i s are pre-existing data, collected and processed by the workflow. During the execution of the workflow, each Seq_i has an external identity (e.g. urn:gb:seq1). This identity is lost as it becomes a text string by step 1, which is subsequently recovered by step 2, but this step also gains each sequence an additional Taverna identity.

different sequence databases and some BLAST services (such as the Position specific iterative BLAST (PSI-BLAST) service) are more sensitive than others. Thus, **WF1** needs to be re-run using different BLAST services to verify that consistent results can be obtained with varying BLAST services.

The data products of these repeated runs of **WF1** are compared to draw conclusions, e.g. an updated sequence was produced in a rerun of **Case 1**, or “**score=100**” is the best parameter setting for running **WF1**. These conclusions need to be justified using the provenance of experiment data products.

2.1 Collected and Computed data

A data product can be either **computed** or **collected**:

- a **computed** data product is generated as a consequence of a workflow execution. $Brpt_0$, $GBrpt_0$, SEQ and $GBRPT_0$ are all generated data products.
- a **collected** data product is one that was pre-existing and has been retrieved from the database over which the BLAST algorithm has been executed. The protein sequences Seq_i s from the sequence database, contained in $Brpt_j$ are an example of this.

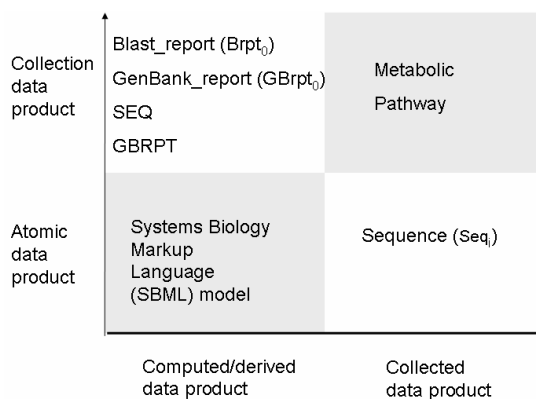


Fig. 2. (a) A data product can be either collected from existing external data sources or computed as a result of a workflow run.

A **computed** or **collected** data product can be either an **atomic** data product or a **collection** data product. Each collection data product contains a collection of *elements*, which are either atomic or collection data products, following the usual recursive composite pattern. Whether a data product is atomic or collection can be rather subtle, dependent on its context, content and how it is used in analyses. For example, a System Biology model [8] data product expressed using the System Biology Markup Language (SBML) [9] is a computed data product, generated as a result of a workflow run, for representing biochemical reaction networks. An SBML model is an atomic **computed** data product. The content of the BLAST report $Brpt_0$ in Figure 1 is a collection of collected data objects Seq_i s. A metabolic pathway [10] is an example of collected from existing

databases and contains a series of chemical reactions occurring within a cell. A collection data product might be computed by the iterations in a workflow run. For example, in Figure 1, SEQ_0 from the `BLAST_simplifier` service, contains a collection of atomic protein sequences Seq_i s; the $GBRPT_0$ contains a collection of atomic GenBank reports $GBrpt_i$ s.

2.2 Using provenance graphs

In ^{my}Grid, for a data product d_i , its data provenance record arising from *each* workflow run r_j forms as a *provenance graph* G retrieved by the pair (d_i, r_j) . The graph provides the context in which each data product d_i was produced. It records

1. d_i 's derivation path:
 - d_i is derived from another data product d_j , either an atomic or a collection. For example, in Figure 1, the BLAST report $Brpt_0$ in this run of WF1 is derived from the input sequence Seq_0 .
 - d_i is derived from a set of parameter settings, $P_i = \{j|p_j, 0 \leq j \leq n\}$, which is used by the service s_h in the run r_i that produced d_i . For example, in Figure 1, the BLAST report $Brpt_0$ in this run of WF1 is derived from the parameter set $P_0 = \{database_0, evaluate_0, score_0\}$.
2. d_i 's membership: d_i (either an atomic or a collection) could be an element of a collection data product d_j . In Figure 1, for a $Seq_i \in SEQ_0$, atomic Seq_i is an element of the collection data product SEQ_0 .

Derivation paths and membership interact, an effect made more complex when iteration over collections is factored in. These details are outside the scope of this paper.

- If d_i is a **collected** data product, the same data object can be collected in another run; e.g. the protein sequence Seq_1 appears as a result in both r_1 and r_2 of WF1. Thus both have Seq_1 *in common*.
- If d_i is a **computed** data product, a data object, d_k that contains the same data value can be computed in another run; e.g. $Brpt_0$ from r_1 and $Brpt_0$ from r_2 contain exactly the same sequences, despite changes in time, parameters or the contents of the external database. The two BLAST reports *correspond*.

Consider a common or corresponding data product, d_i , appearing in two runs, r_1, r_2 . Analysis of the provenance of d_i becomes: (a) for audit, a provenance graph G_i ; (b) for aggregation, the retrieval of G_1 and G_2 ; (c) for integration, the merge of G_1 and G_2 ; (d) for comparison, the difference between G_1 and G_2 .

2.3 Data identity

In order to *aggregate* the provenance of a d_i computed or collected in multiple runs, we need to identify this d_i produced in each run. In order to *integrate* and *compare* the multiple provenance graphs of a common data product d_i produced in multiple runs, we need to identify all the common or corresponding data products and parameter settings recorded in these graphs. Provenance is

represented by RDF in ^{my}Grid, which is a graph of data products and control parameters connected by their provenance relationships. Each data product and control parameter is identified by a URI in the RDF provenance graphs. We need to manage the identities to make sure that common or corresponding data products and parameters are identified uniquely across workflow runs. In the next section we explain the identity issues for WF1.

3 Identities

In Figure 1, all identities of all data products collected, or computed by a workflow are managed by the LSID protocol. An LSID is a URI that is standardized by the OMG LSR group ⁵. An LSID consists of five parts separated by colons: a prefix (urn:lsid); the authority name (mygrid.ac.uk); the authority-specific data namespace (data); the namespace-specific object identifier (49841) and a version number of the object (1) leading to a URI: `urn:lsid:mygrid.ac.uk:data:49841:1`. An LSID authority for a resource allocates an identity and resolves it for the resources for which it is an authority (and no others), guaranteeing that the data is immutable. Each data resource has a responsibility for managing its own LSID authority. Even when resources are replicated locally a different, local LSID authority is in place.

Consider the identities allocated for the data products shown in Figure 1 in one run of the workflow WF1,

- every **collected** data product, the protein sequence Seq_i collected at step 2, carries its **externally** allocated identity with it. The **external** identity for Seq_i is contained within the content of the data product, e.g. `urn:gb:seq1` for Seq_1 . A collection collected data product (not shown in Figure 1), e.g. a pathway data product, might also carry an **external** identity.
- every **computed** data product is given a Taverna identity by the Taverna workflow system, e.g. `urn:tav:b0` for $Brpt_0$ ⁶.
- every collection data product **computed** by iterations, for example $GBPT_0$ and SEQ_0 , is allocated with a Taverna identity, e.g. `urn:tav:g0` for $GBPT_0$ and `urn:tav:seq0` for SEQ_0 .

A **collected** data product may carry an **external** identity, but a **computed** data product is assigned with a Taverna identity. This impacts on the protocol for managing data identities as shown in Section 4 and the data typing system in Taverna, as discussed in Section 6. Using this workflow, we now explore *external* identity allocation by data resources and *internal* identity allocation by Taverna.

3.1 External identity: Resource generated identities

At least 700 different, heterogeneous resources are available in life sciences [11]. The autonomy of data providers enables rapid generation and deployment of

⁵ <http://www.omg.org/cgi-bin/doc?lifesci/2003-12-02>

⁶ These are LSIDs. We abbreviate the representation of an LSID throughout this paper.

new resources, but they rarely conform to any community-wide standards, often allocating different identities for a common data object. We find the following situations:

- equivalent data objects in different databases. For example, “gi:15145617” (GenBank⁷) and “ac073846” (EMBL-Bank⁸) are the same DNA sequence. The protein “Dual specificity DE phosphatase Cdc25C” is identified as “aaa35666” in GenBank and “p30307” in UniProt⁹.
- equivalent data objects in different replicas is a variation of the previous point. A resource is often replicated remotely or locally, and sometimes locally extended or customised. For example, the same sequence is “urn:lsid:myg:ac073846” for a local copy of EMBL-Bank in ^{my}Grid with its own LSID authority.
- equivalent data objects from different workflows. The invocation of a Kyoto Encyclopedia of Genes and Genomes (KEGG¹⁰) pathway service (instead of a BLAST service) produces a pathway result containing a collection of protein sequences. Some of the sequences in the *Brpt*₀ data product shown in Figure 1, also appear in the pathway data product, but now with KEGG LSIDs.

These arrangements result in duplicate external identities for equivalent data products collected from different databases, replicas or by different services. For example, the sequences *Seqis* collected from the external database by the BLAST service in Figure 1.

3.2 Taverna Identity: Workflow generated identities

Figure 3 gives the ^{my}Grid LSID allocation architecture. In a workflow run, when a data product is passed to the enactor, it is allocated with a Taverna LSID by the Taverna LSID authority. This identity is associated with the data product when it is stored or passed to invoke other services by the enactor. The data products acquired (collected or computed) by a workflow run are stored in a customized database as part of the workflow process or a local “catch all store”, the relational data store BAKLAVA. This identity is also used to store the provenance metadata of this data product in the metadata store, KAVE. Data and provenance are immutable in ^{my}Grid. Once data products and provenance are preserved in either the BAKLAVA/customized database or the KAVE, they should not be deleted or altered. Provenance of a data product can be augmented with more provenance metadata. A client communicates to the ^{my}Grid data and provenance repositories over the network by the LSID protocol to retrieve data or metadata of a data product by its LSID.

Migration duplicate identity Data products generated by Taverna and stored in a customized database or the BAKLAVA store can be archived or replicated

⁷ <http://www.ncbi.nlm.nih.gov/Genbank/>

⁸ <http://www.ebi.ac.uk/embl/>

⁹ <http://www.ebi.uniprot.org/>

¹⁰ <http://www.genome.jp/kegg/>

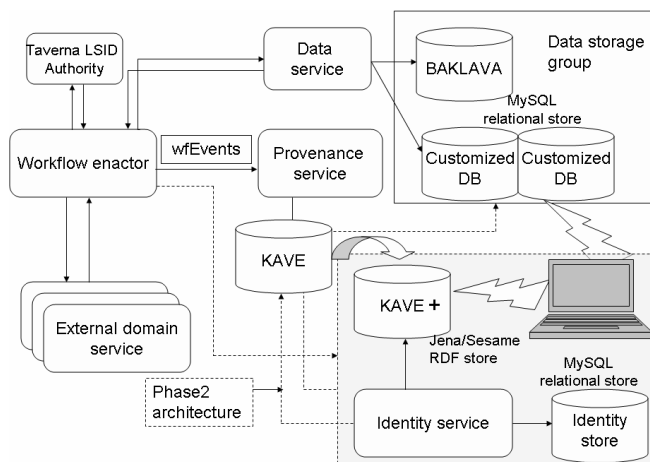


Fig. 3. ^{my}Grid’s architecture for allocating LSIDs for data products during workflow executions. The dashed part represents the extended architecture when introducing the identity service to manage the identities for common or corresponding data products.

among scientists in their own file stores. A migration duplicate is caused by the failure to migrate data identities with the data when the data are curated. For instance, when a data product is copied from the BAKLAVA store to a personal file system, its identity is deprecated and replaced with a new identity such as a path to access the file system.

Execution duplicate identity Workflows are repeatedly run to collect updated data products as the underlying data resources are updated. In each independent run, the Taverna LSID authority is ignorant of the existence of common or corresponding data products produced in different runs and the existence of the duplicate identities allocated for these data products. Therefore, duplicate identities for common or corresponding data products are produced in repeated executions:

1. corresponding computed data products. Two BLAST reports can be generated in two executions of WF1 that contain exactly the same protein sequence objects. The reports are identified by different Taverna identities because they are from different runs and are, in fact, different reports. When we use provenance graphs from two runs that have corresponding reports, we would like to assert that they are “equivalent” within an experimental context.
2. common collected data products. As discussed previously, we would like to identify data products that provenance graphs have in common. Two runs, r_1 and r_2 , collect the same data product d_0 (in the context of Figure 1 this could be Seq_i); thus two provenance graphs share this common protein sequence (Figure 4(a)). To recognize that the data product is in common we assert that the two data objects produced by the workflows are “equivalent”; in fact they are the same object. The most obvious mechanism is that they should have the same identifier. However, the processing of the workflows means

that this is not the case. In this paper *equivalent* data products include both collected products in common and corresponding products that are computed.

- When step 1 is finished, the computed data product $Brpt_0$ is stored and allocated with a Taverna LSID `urn:tav:b0`. The collected data objects, i.e. the protein sequences Seq_i s, contained within are neither extracted nor allocated with any LSIDs. This is because they have been turned into text by BLAST and their external identities, e.g. `urn:gb:seq1`, are contained in their data contents as strings.
- When step 2 is finished, each sequence, such as Seq_1 , has been extracted and stored. Because it is a new object, recovered by post-processing, it is automatically allocated with a Taverna LSID `urn:tav:seq1`, despite the fact it already has an external LSID that it carried. In each workflow run of WF1, if Seq_1 appears it is given a new, different Taverna LSID. Thus the same data product has its external LSID and, for each run, a Taverna LSID.

Two cases further compound the problem:

1. corresponding collection data products *computed* in iterations. For instance, the data product from the `GenBank.retrieve` service in Figure 1 produced in one run of WF1, i.e. $GBRPT_0$ contains a collection of GenBank reports $GBrpt_i$ s. This collection data product and its elements are always allocated with new, different Taverna identities each time they were produced. In our identity service, we define a *correspondence* operation to decide two collection data products produced by iterations are corresponding to each other.
2. equivalent nested data objects in a data product. For instance, each sequence Seq_i contains some nested data objects such as the species data object. These nested data objects are allocated with new, different Taverna identities each time they are extracted.

Duplicates are not only due to inadequate attention to our identity allocation mechanism, but also due to the following:

- (a) differentiating the data product and its data derivation path produced in one context with its common or corresponding data product produced in another context. Figure 4(a) shows the two derivation paths for the data product d_0 that were collected in different runs. If during the provenance collection this d_0 was identified by the same identity, as shown in Figure 4(b), only the merged derivation path will be kept in KAVE. It becomes difficult to retrieve “the data product which d_0 was derived from that was produced in run r_1 ”. myGrid tries to solve this problem by incorporating more context information with a data product using Named Graphs [12]. The details are beyond the scope of this paper.
- (b) potential computation costs to avoid publishing duplicate identities at run time for common or corresponding data products. The correspondence between computed data products is based on their values most of the time, which is inefficient. For instance, the identity correspondence of our example BLAST reports cannot be decided based on the identities of the sequence data objects contained in these data products, as the same sequence dupli-

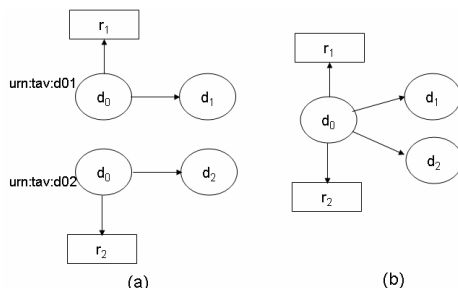


Fig. 4. (a) d_0 identified as `urn:tav:d01` was derived from d_1 in the run r_1 and its equivalent data product d_0 identified as `urn:tav:d02` was derived from a different data product d_2 in the run r_2 . (b) If d_0 is identified by one identity during the provenance collection in ^{my}Grid, the data derivation paths for d_0 that were generated in different runs will be converged.

cate identities are published by different databases. Evaluating the equivalence or correspondence of data products at run time could slow down the workflow enactor, but could be achieved by a post workflow enactment process.

4 Identity solutions

In order to cope with the `execution` identity problem we propose an identity protocol for building co-references of execution duplicates, and how we might revise Taverna’s identity naming scheme. This protocol is asynchronous to workflow enactment and data and provenance collection. We ignore the external duplicates in this protocol.

4.1 Co-reference identity protocol

Our solution is based upon an `IDSet` entity that stores co-references of duplicate identities for equivalent data products, by adopting the work from Lewy *et al* [13]. Consider Figure 1, $IDSet_1 = \{Brpt_0, \{\text{urn:tav:b0}\}, \{\text{urn:tav:b3}\}\}$ is an `IDSet` for the BLAST report $Brpt_0$: each `IDSet` object has its own identity $IDSet_1$, is associated with the content of a data product and contains a set of duplicate identities for this data product. These `IDSet` objects are stored in the identity store by the identity service, shown in Figure 3. In the following, we abbreviate an `IDSet` as: $IDSet_1(Brpt_0) = \{\{\text{urn:tav:b0}\}, \{\text{urn:tav:b3}\}\}$.

The benefits of this approach are the following:

1. the co-references for a data product can be queried using one of its identities, e.g. `urn:tav:b0` can be used to find $IDSet_1$ and then to retrieve all the co-references held in $IDSet_1$.
2. the co-references for a data product can be queried using its content, e.g. the value of $Brpt_0$ can be used to find $IDSet_1$.
3. the `IDSets` for a data product that are in different identity stores maintained by different users or groups can be merged, e.g. $IDSet_3(Brpt_3) = \{\text{urn:tav:}$

$\mathfrak{b3}$ }, created for $Brpt_3$, can be merged with $IDSet_1$ associated with $BRpt_0$, the correspondence to $BRpt_3$.

The dashed part in Figure 3 shows the architecture of how this protocol (a) functions when executing a workflow in Taverna and (b) interacts with other components in ^{my}Grid. Three actors participate in the identity protocol: the external service, the workflow enactor and an identity service.

1. the workflow enactor passes data to external services then invokes these services based on the workflow design.
2. the service returns the data results and the external identities for collected data products. When the service returns `collected` data products in a response message to the enactor, it should publish identities for these data products in its response message. If the service failed to do this, it would be regarded as a service with lower quality than those that do so.
3. when the data products are returned to the enactor by the service, the enactor assigns Taverna LSIDs to these data products.
4. the enactor invokes the identity service, passing messages to the identity service that includes: the data products, their Taverna LSIDs and associated external LSIDs (if any).
5. (1) the identity service (a) intercepts Taverna identities for collected and computed data products and (b) retrieves or builds `IDSet` objects for these data.
 - (2) the identity service updates the `IDSet` objects in the *identity* store and inserts these `IDSet` identities as metadata of the data products into the *provenance* store KAVE.
 - (3) the enactor stores the data products in the data store BAKLAVA and their provenance in the provenance store KAVE.

The step of building `IDSet` objects can be taken offline, as shown in Figure 3, by analyzing the provenance store and the data store after the workflow run. The relational identity store is used for keeping the `IDSet` objects. The RDF KAVE store contains provenance metadata as well as the `IDSet` metadata for the data products. We name this upgraded KAVE as *KAVE+* in the **Phase 2** architecture. These `IDSet` metadata are generated by the following scheme and can be used when integrating and comparing provenance graphs, as shown in the Section 4.3.

4.2 Revised identity naming scheme

This identity protocol constructs and updates `IDSet` objects by three means: (1) identities of `collected` data products, (2) data values of `computed` data products and (3) data objects contained in collection data products `computed` by iterations.

Allocation1. Allocation by identities The identity service receives a `collected` data product, either an atomic, e.g. a *Seq*; or a collection, e.g. a pathway, which collects a set of external data objects:

1. search for an existing IDSet object using the external identity of this data product. If an IDSet is found, retrieve it; otherwise create one.
2. update the identity store with the data product's external and Taverna identities, and its IDSet object identity.
3. insert this IDSet object and its relationship with the data product as RDF statements into the provenance metadata store KAVE+.

Allocation2. Allocation by values The identity service receives a **computed** data product, either an atomic one or a collection, e.g. *Brpt*, which collects a set of external data objects:

1. search for an existing IDSet object using the data product's data value. If an IDSet is found, retrieve it; otherwise create one.
2. update the identity store with the Taverna identity of this computed data product and the IDSet object identity.
3. insert this IDSet object and its relationship with the data product as RDF statements into the provenance.

Allocation3. Allocation by objects The identity service receives a collection **computed** data product generated by an iteration, which contains a collection of atomic or collection data products. This data product and each of its elements is identified by a Taverna LSID, e.g. in Figure 1 the SEQ_0 which contains a collection of atomic **collected** data products Seq_i , or the the $GBRPT_0$ which contains a collection of atomic **computed** data products $GBrpt_i$. The *correspondence* operation for a collection data product produced by iterations is defined as: two collection data products D_i and D_j generated by iterations correspond, if and only if both collections have the same size, and all elements in the two collections are equal. For a collection data product D_i , the identity service should:

1. search for any existing IDSet objects for each element of D_i . Search by the element's identity if it is a **collected** data product; and search by the element's value if it is a **computed** data product.
2. search for its corresponding collection data product and an existing IDSet object. If an IDSet is found, retrieve it; otherwise create one.
3. update the identity store with D_i 's Taverna identity, and its IDSet object identity.
4. insert the relationship of the collection data product and its element data products into the identity store.
5. insert this IDSet object and its relationship with the data product D_i as RDF statements into the provenance metadata store.

This protocol manages only the internal duplicate Taverna identities for equivalent data products. Currently, duplicate identities continue to be allocated in ^{my}Grid to avoid the costs of allocating unique identities for equivalent data products during workflow runs. This protocol and naming scheme, however, improves the maintenance of: (a) the external identities associated with collected data products; (b) the relationship between computed data product

and its element computed or collected data products; and (c) the internal identity co-references.

4.3 Putting the Identity Service to Use

An identity service prototype was implemented as a plug-in to Taverna, building IDSet objects for equivalent data products during workflow runs, as shown in Figure 3:

- For the **collected** data products, such as the protein sequences Seq_i s, their external identities are parsed and extracted from the data contents. The IDSet objects for them are built by these external identities.
- For the **computed** data product that contains a collection of external data objects, such as the $Brpt_0$, their IDSet objects are built by the identities of the contained external data objects. This is to avoid the cost of storing and sorting big blobs of data in the MySQL identity store in this prototype.
- For the **collection** data products **computed** in iterations, such as the SEQ_0 , $GBRPT_0$ in Figure 1, their IDSet objects are built by the identities of their element data products, either atomic or collection.

We conducted a preliminary assessment on how the identity service can help to achieve the goal of integrating provenance graphs from repeated runs of the workflow **WF1**. The provenance of repeated runs of workflow **WF1** were collected, and identity co-references between equivalent data products were built using the *identity service*. For every two runs of **WF1**, in order to integrate provenance of repeated runs, we conduct the following:

1. aggregate the data provenance graphs of 2 runs of **WF1**. This returns two provenance graphs G_1 and G_2 .
2. normalize each provenance graph G_i . For each data product in the provenance graph, retrieve its IDSet identity. If an IDSet identity is found, replace the data identity in the provenance graph with the IDSet identity. This operation returns a normalized graph, G'_i , with each equivalent data product produced in different runs being identified by its IDSet identity.
3. compare the normalized graph G'_1 and G'_2 .

We succeeded in detecting the similarities and differences between one provenance graph generated in one run of **WF1** with another graph generated in another run. This approach is much faster than building the co-references between equivalent data products at the time of comparison.

The number of data objects contained in each provenance graph decides the size of a provenance graph. The computational complexity of normalizing a provenance graph is decided by the size of this graph. An optimization of this normalization is needed if a large provenance graph is to be processed.

5 Related work

Duplicate identities are a common problem in data integration. RefSeq [14] builds cross-references for sequence data across multiple major sequence databases. The

Handle system provides GUIDs for digital objects assured by a global naming authority [15]. This is hard to achieve in life sciences because of the autonomy of data and tool providers. Our identity protocol focuses on constructing co-references between internal duplicates for equivalent data products. These identity co-references can be published at different places and then merged by the set calculus. This identity protocol can be adopted in many service-oriented architecture such as the provenance collection architecture proposed by Groth [16].

The Virtual Data Language (VDL) [17] represents derivation relationships between data that were processed by computational procedures. It is unclear whether the aggregation or integration of provenance over repeated reruns of the same workflow over the same data is an issue. The focus is on computed data products and the identity issue becomes one of data mining for identical data values rather than the maintenance of external data object identity and taking care not to allocate duplicates.

If unique identities are allocated for equivalent data products produced in repeated runs, a mechanism is required to differentiate the different contexts in which each data product was produced. The PASOA project includes a workflow run id as part of the data product id that is produced in a particular workflow run [16]. However, additional contextual information is needed by bioinformaticians, such as the person who produced the data, and an intermediate data product from which a collection of data products were derived from. ^{my}Grid adopts Named Graphs [12] to incorporate more contextual information with data products.

6 Conclusion

In previous work we focused on building a provenance model and the technology to represent this model [2]. The model was developed to assist not only in keeping audit trails of a single workflow run, and the provenance graph of any data product acquired during the run, but also to support the analysis of multiple provenance logs across multiple workflow runs. This analysis requires a means of identifying equivalent data products; both common collected products and corresponding computed products. Allocating and assigning data product identity proved harder than anticipated in practice for provenance graphs that are independently generated, yet need to be combined. Our previous identity allocation scheme works well when we do not aggregate, integrate and compare provenance, but raises issues when we do.

Bioinformatics workflows not only generate new data, but also discover new information by combining and collating existing data. These pre-existing data, external to the Taverna world, have their own identities allocated. In addition, local identities are published for these data products, such as Taverna LSIDs and VDL's Logical File Names. The autonomous nature of the current bioinformatics domain makes it hard to adopt a global naming service unless a community-wide agreement is achieved, such as the data transfer standard in earth science [18] and in the caBIG project (<https://cabig.nci.nih.gov/>). Our problem is particu-

larly acute in that we do not prescribe a closed, strongly typed data environment such as that dictated by caBIG. The multiple identity problem is present in a large extent such as on the Web. Duplicate identities point to the same entity hosted by different web sites. In this paper we tried to enumerate the different identity duplication problems for equivalent data products, and present our solution for execution duplicates, which revises our identity allocation scheme and constructs co-references between duplicate identities.

Whether two data products are equivalent is dependent on the application purpose. In programming languages, the equivalence of two objects can be based on value comparisons or reference comparisons depending on the data structure of the objects. In our identity service implementation the identity equivalence between two data products is decided by reference comparisons. Different equivalence can be customized by different applications. We have proposed value comparisons in the protocol for computed data products. However, this is expensive and needs support by using specialized services.

Two scalability issues remain to be solved: (1) the scalability of identifying corresponding collection data products computed in iterations in different runs. Iterations over iterations in a workflow run produce collections with multiple hierarchies, i.e. collections containing collections iteratively. This makes it expensive to retrieve corresponding collections with multiple hierarchies; and (2) the scalability of exploiting identities. As shown by the example of exploiting identities in Section 4.3, the computation complexity of normalizing a provenance graph is decided by the size of the provenance graph. This normalization is required in provenance integration and comparison. An optimization is needed if a large provenance graph is to be normalized.

The problem of migration duplicates requires *finer-grained* provenance collection. The data derivation collected in ^{my}Grid data provenance is coarse-grained provenance. The relationship between the computed data product and its collected data objects, e.g. the relationship between a *Brpt* and the sequences *Seq_is* shown in Figure 1, and the relationship between a data product and its migrated replicas are finer-grained provenance. VDL can express finer-grained data relationships. This finer-grained data relationship also relates to the PASOA project, which expresses such relationships using an OWL ontology¹¹.

7 Acknowledgements

The ^{my}Grid project, grant numbers GR/R67743, EP/D044324/1 and EP/C536444/1, is funded under the UK e-Science programme by the EPSRC. The authors would like to acknowledge the other members of the ^{my}Grid team for their contributions, and in particular acknowledge Tom Oinn, Matthew Pocock, Daniele Turi and Chris Wroe. We thank Stian Soiland and David Withers for their useful comments.

¹¹ <http://twiki.pasoa.ecs.soton.ac.uk/bin/view/PASOA/Ontologies>

References

1. Stevens, R., Tipney, H.J., Wroe, C., Oinn, T., Senger, M., Lord, P., Goble, C., Brass, A., Tassabehji, M.: Exploring Williams-Beuren Syndrome Using ^{my}Grid. *Bioinformatics* **20** (2004) i303–i310
2. Zhao, J., Wroe, C., Goble, C., Stevens, R., Quan, D., Greenwood, M.: Using semantic web technologies for representing e-science provenance. In: Proc. of the Third International Semantic Web Conference. Volume 3298., Hiroshima, Japan (2004) 92–106
3. Li, P., Hayward, K., Jennings, C., Owen, K., Oinn, T., Stevens, R., Pearce, S., Wipat, A.: Association of variations on i kappa b-epsilon with graves' disease using classical and mygrid methodologies. In: Proc. UK e-Science All Hands Meeting. (2004)
4. Oinn, T., Greenwood, M., Addis, M., Ferris, J., Glover, K., Goble, C., Hull, D., Marvin, D., Li, P., Lord, P., Pocock, M.R., Senger, M., Wipat, A., Wroe, C.: Taverna: Lessons in creating a workflow environment for the life sciences. *Journal of Concurrency and Computation: Practice and Experience* (2005) in press.
5. Clark, T., Martin, S., Liefeld, T.: Globally distributed object identification for biological knowledgebases. *Briefings in Bioinformatics* **5** (2004) 59–70
6. Dalziel, J.: DOI in a DRM environment. white paper, Macquarie University (2004)
7. Altschul, S., Gish, W., Miller, M., Myers, E., Lipman, D.: Basic local alignment search tool. *Journal of Molecular Biology* **215** (1990) 403–410
8. Li, P., Pinney, J.W., Oinn, T., Knight, C., Kell, D.B.: Experiences in automating the construction of biological models in sbml using the taverna workflow system. In: BBSRC poster. (2005)
9. Hucka, M., Finney, A.: Escalating model sizes and complexities call for standardized forms of representation. *Molecular Systems Biology* **1** (2005)
10. Ruttenberg, A., Rees, J.A., Luciano, J.S.: Experience using owl dl for the exchange of biological pathway information. In: Proc. Internal Semantic Web Conference, OWLWorkshop. (2005)
11. Galperin, M.Y.: The molecular biology database collection: 2006 update. *Nucl. Acids Res.* **34** (2006) 3–5
12. Bizer, C., Carroll, J.: Modelling context using named graphs. In: SWIG meeting. (2004) unpublished.
13. Lewy, T., Glaser, H., Shadbolt, N.: A Framework for Reference Management in the Semantic Web. Technical report, (University of Southampton)
14. Pruitt, K.D., Maglott, D.R.: Refseq and locuslink: Ncbi gene-centered resources. *Nucleic Acids Research* **29** (2001) 137–140
15. Kahn, R., Wilensky, R.: A framework for distributed digital object services. Technical Report tn95-01, Macquarie University (1995)
16. Groth, P.T., Luck, M., Moreau, L.: A protocol for recording provenance in service-oriented grids. In: Proc. of the Eighth International Conference on Principles of Distributed Systems, Grenoble, France (2004) 124–139
17. Foster, I., Vockler, J., Wilde, M., Zhao, Y.: The virtual data grid: A new model and architecture for data-intensive collaboration. In: First Biennial Conference on Innovative Data System Research. (2003)
18. Arctur, D.K., Hair, D., Timson, G., Martin, E.P., Fegeas, R.: Issues and prospects for the next generation of the spatial data transfer standard (SDTS). *International Journal of Geographical Information Science* **12** (1998) 403 – 425