# Traceability Mechanisms for Bioinformatics Scientific Workflows

**Luciano A. Digiampietri**
Institute of Computing, University of Campinas
CP 6176, 13084-971, Campinas, SP, Brazil
e-mail: luciano@ic.unicamp.br

**Claudia Bauzer Medeiros**
Institute of Computing, University of Campinas
CP 6176, 13084-971, Campinas, SP, Brazil
e-mail: cmbm@ic.unicamp.br

**João C. Setubal**
VBI, Virginia Tech, Bioinformatics 1
Box 0477, Blacksburg, VA, USA

**Roger S. Barga**
Microsoft Research, One Microsoft Way
Redmond, WA 98052, USA

**Abstract**

*Traceability* and *Provenance* are often used interchangeably in eScience, being associated with the need scientists have to document their experiments, and so allow experiments to be checked and reproduced by others. These terms have, however, different meanings: provenance is more often associated with data origins, whereas traceability concerns the interlinking and execution of processes. This paper proposes a set of mechanisms to deal with this last aspect, the solution is based on database research combined with scientific workflows, plus domain-specific knowledge stored in ontology structures. This meets a need from bioinformatics laboratories, where the majority of computer systems do not support traceability facilities. These mechanisms have been implemented in a prototype, and an example using the genome assembly problem is given.

## Introduction

This paper is concerned with combining research on databases and on scientific workflows applied to the development of a framework that seamlessly supports the backwards and forward traceability of bioinformatics experiments, at distinct abstraction levels. The term *traceability* is used here to denote the ability to describe the way in which some product has been developed, including all processes it went through. *Provenance*, on the other hand, is associated with a place or origin, and is usually linked to data i.e., when, where and who produced it. Provenance normally does not demand a detailed process trace. Both issues – traceability of processes and provenance of data – are very important in any kind of scientific experiment.

In bioinformatics, for instance, the quality of an experiment depends heavily on properly identifying both data origins and the processes that produced these data (Buttler *et al.* 2002). However, it is common practice that provenance is captured by laborious manual annotations, which often vary across laboratories; moreover, the majority of computer systems in a laboratory do not provide traceability mechanisms. Our work concerns traceability, and thus the ability to "discover the cause or origin of something by examining the way in which it has developed" (Cambridge ).



Different groups of scientists have distinct needs in terms of provenance and traceability mechanisms (Barga & Digiampietri 2006; 2007). To cope with this issue, we propose a layered and flexible solution to deal with the range of user needs. Furthermore, a bioinformatics framework must contain integration mechanisms to provide a transparent access to heterogeneous data and to provide interoperability among tools and systems.

Thus, the paper is concerned with how to record detailed information about all processes involved in an experiment to produce a given result, such as an assembled genome or a phylogenetic tree. Our solution is based on combining annotated scientific workflows with database mechanisms. Annotations and workflows are stored in a database, which allows supporting traceability requirements through database queries, allowing for both forward and backward tracking within a bioinformatics experiment. We are testing the framework for bioinformatics problems in the tasks of genome assembly.

The main contributions of this paper include the following: (1) Description of a framework to support the tracing of all steps of a bioinformatics experiment, taking advantage of standard database storage and querying mechanisms; (2) Extension of database query facilities to support domain-specific concerns (e.g., in the bioinformatics context, allowing queries concerning alignment, base-calling, etc); (3) Validation of the proposed approach via a prototype, tested using real laboratory data.

The remainder of this paper is organized as follows. Section Main Concepts and Related Work describes basic concepts and related work. Section Architecture presents our architecture. Section Implementation Issues presents examples where our system is helping scientists. Section Conclusions contains our conclusions and plans for future work.

## Main Concepts and Related Work

This section discusses concepts used in the paper and reviews related work, focusing on workflow and provenance representation, data integration and interoperability, and traceability. Finally, we describe our previous work whose contributions include mechanisms to facilitate bioinformatics workflow composition (Digiampietri *et al.* 2006; Digiampietri, Pérez-Alcázar, & Medeiros 2007), and efficient storage of provenance information (Barga & Digiampi-

etri 2006; 2007).

## Basic Concepts

The *genome assembly* problem consists in joining and matching together pieces of DNA sequences to create a contiguous sequence, much in the way jigsaw puzzles are put together. Fragment sequences are created inside a laboratory by procedures that extract pieces from a species' DNA and then produce long strings of base pairs (ACGT). The main challenge in this process is finding the appropriate means of assembling the fragments together into a biologically correct sequence. For more details on bioinformatics problems see (Setubal & Meidanis 1997).

One of the challenges in genome assembly is how to combine the several complex activities that are used in these processes. These activities use heterogeneous data from distinct sites and need a high degree of human intervention. *Scientific workflows* are being used as a means to design and execute these complex activities (Oinn *et al.* 2004). A *scientific workflow* (Wainer *et al.* 1996) is the specification of a process that describes a scientific experiment. Such workflows allow the representation and support of complex tasks that use heterogeneous data and software (Cavalcanti *et al.* 2005). In particular, in bioinformatics they are characterized by variability in workflow design for the same task. There are several open problems involved in scientific workflow construction, sharing and validation. Among them, the paper is concerned with traceability mechanisms.

## Related Work

Scientific workflows (Wainer *et al.* 1996) are being increasingly adopted as a means to specify and coordinate the execution of experiments that involve participants in distinct sites. Such workflows allow the representation and support of complex tasks that use heterogeneous data and software (Cavalcanti *et al.* 2005). Bioinformatics is one of the research domains that has widely profited from this facility (Hoon *et al.* 2004; Oinn *et al.* 2004; Stevens *et al.* 2004). However, these systems do not capture provenance information or provide the necessary features to exploit it, hampering the validation and reuse of bioinformatics data.

**Workflow and Provenance Representation.** The Workflow Management Coalition (WFMC) (Hollingsworth 1995) has defined a generic model for workflow representation to provide interoperability among different workflow systems. This model has been extended in several ways - e.g. to provide more semantic information or facilitate the sharing of scientific experiments. Also to helps interoperability, many tools invoked by such workflows are now encapsulated into Web services.

In particular, we take advantage of the extension described in (Medeiros *et al.* 2005; Pastorello Jr. *et al.* 2005), which concerns a multi-layered workflow representation to support sharing and reuse at different abstraction levels. Workflows are represented in three layers: abstract workflow (information about the generic structure of a workflow), concrete workflow (instances of the abstract workflow) and executable workflow (a workflow with its input data and parameters).

Several groups have conducted research on provenance representation (Moreau & Foster 2006; Pastorello Jr. *et al.* 2005). A provenance model must represent the "what", "when" and "where" of information managed in a system. We argue that a provenance model must also be flexible to address distinct levels of detail in workflow representation, and to consider the needs of distinct user groups. Some of our previous work concerns a layered provenance model that considers such flexibility (Barga & Digiampietri 2006; 2007).

Our workflow model combines the features of the layered workflow model (Medeiros *et al.* 2005) and the layered provenance model (Barga & Digiampietri 2006; 2007). This combination allows workflow and provenance information to be stored, queried and shared at different abstraction layers. We also added a fourth layer called *runtime workflow* to store information about the executions of an *executable workflow*.

**Data Integration and Interoperability.** There are several approaches to deal with data heterogeneity, varying from the conceptual to the physical level (Hernandez & Kambhampati 2004). All of these approaches require establishing a mapping among different perspectives of the world. The conceptual level perspective provides to all users a single conceptual model of their applications. This approach has been widely used in bioinformatics frameworks (Peterson *et al.* 2001; Stein *et al.* 2002; Diehn *et al.* 2003); this kind of solution is useful when the groups involved use the same model, but it does hamper data sharing among groups that use different models. At the other end of the spectrum lies the approach involving structure (and conceptual) mapping (Bowers & Ludäscher 2004). This approach is based on keeping a record of all data transformations that are needed in a system to allow the integration of heterogeneous data from several sources. This record is then updated whenever a new kind of transformation is available.

Our framework is unique in that it uses a structure mapping record combined with ontological description of the concepts to provide a flexible way to integrate heterogeneous data. This combination takes advantage of the relationships present in the ontologies to map concepts into each other, when defined by distinct user groups.

Global conceptual models or structure mapping are common solutions for data integration. Interoperability among systems requires additional approaches, including data exchange standards, use of Web services and interface matching algorithms (Hernandez & Kambhampati 2004). The latter are based on trying to match a request for some data or process (caller request) with candidates that best fit this request. Approaches vary according to the compatibility between interfaces – e.g., (Santanchè & Medeiros 2005).

Our proposal combines the matching and ranking algorithm from Santanchè and Medeiros (Santanchè & Medeiros 2005) with the structure mapping of Bowers and Ludäscher (Bowers & Ludäscher 2004), enhanced with workflow composition mechanisms (Digiampietri *et al.*

2006; Digiampietri, Pérez-Alcázar, & Medeiros 2007). The goal is to facilitate the integration of data and the interoperability among different sites.

**Traceability.** Traceability mechanisms, in the sense used in this paper, are typically found in work that deals with software engineering, electronic commerce or supply chain problems. In software engineering, traceability is used to establish relationships among requirements, specifications, design and the corresponding code (Maletic, Collard, & Simoes 2005). This allows the verification of, for instance, how requirements evolve to an implemented functionality or to start from a piece of software and trace back the corresponding requirements. In electronic commerce systems, traceability is considered a trust enhancer that is used to trace transactions, payments, and measurements, which provides the assurance of fairness and methods to establish legal proof and redress (Steinauer, Wakid, & Rasberry 1997). In supply chain systems, traceability means providing the identification of all the steps (processes) that were executed to generate a given product, including all the raw matter and intermediate inputs used, the quality of each input, the place where it was produced, etc (Opara 2003).

Our proposal aims to take advantage of these traceability mechanisms and apply them to bioinformatics, allowing the back- and forward trace of experiments.

## Architecture

We have developed a framework (Digiampietri *et al.* 2006; Digiampietri, Pérez-Alcázar, & Medeiros 2007) to take advantage of ontologies to support the specification and annotation of bioinformatics workflows. Our framework extends Artificial Intelligence planning techniques with ontologies to support design, reuse and annotation of bioinformatics experiments, specified as scientific workflows. Each activity within such a workflow can be executed either by invocation of a Web service or of another (sub-)workflow. Workflows are stored in databases, and ontology repositories are used to enhance the semantics of automatic bioinformatics workflow construction.

This paper extends this design and management framework with mechanisms to support full traceability of experiments. This approach takes advantage of a workflow provenance model (Barga & Digiampietri 2006; 2007) to support efficient storage of provenance information.

Figure 1 shows the resulting architecture, portrayed in four layers: Interface, Processing, Data Manager and Repositories. The modules in gray correspond to extensions introduced to support traceability. The Interface Layer contains the graphical interface that presents to the user all the functionalities of the framework. The Processing Layer is composed of several processing modules that are responsible for: register and discovery of services, design of workflows (including their automatic composition) and workflow execution. The Data Manager Layer is the middleware between the repositories and the rest of the system. It contains features to support interoperability, data integration and traceability. The Repositories Layer stores information about services, structure types, ontologies, data transformation rules, workflows and provenance data.

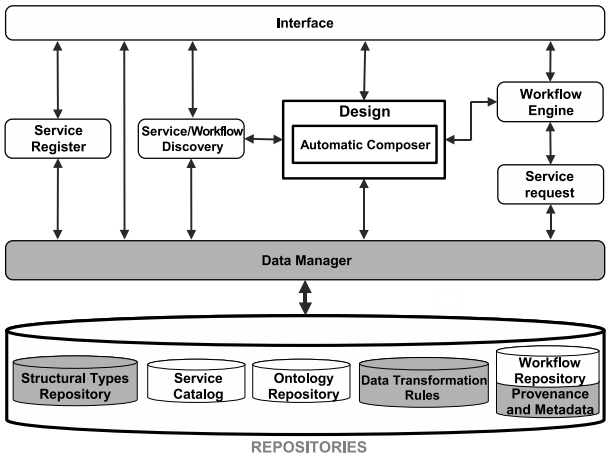


Figure 1: System Architecture - gray boxes show features to support traceability

### Repositories

Five data repositories, stored in a database, serve as the basis for the design, annotation, execution, sharing and traceability of bioinformatics experiments: *Ontology*, *Structural Types*, *Service Catalog*, *Workflow* and *Data Transformation Rules*.

The Ontology Repository contains information about concepts and service types of a given domain. The Structural Types Repository records information about the ways in which the instances of a concept can be stored. The Workflow Repository stores workflows (and parts thereof), in the four abstraction layers described in Section Workflow and Provenance Representation, as well as all data needed to run these workflows. The Workflow Repository also has now been extended to contain provenance information and metadata. The Data Transformation Rules Repository stores information about the valid transformations from a data structure to another.

In more detail, the *Ontology Repository* stores two kinds of ontologies: *Domain Ontology* and *Service Ontology*. The Domain Ontology contains the relationships among the concepts of an application domain (in our case, bioinformatics concepts – e.g., *genome*). The Service Ontology stores the types of services used in this domain, without their instantiation – e.g., *alignment service* or *base–calling service*.

The *Structural Types Repository* specifies the data structures for storing data used in an experiment – e.g., the inputs, outputs, parameters of an experiment. Structural types can be basic (integers, strings, etc) or, recursively, a composition of types.

The *Service Catalog* extends the role of an UDDI (Universal Description, Discovery, and Integration). It stores information about service providers, the Web services available and their interfaces. Furthermore, it also stores non-functional attributes of the service instances, qualifying service interfaces with terms from our Domain Ontology and

classifying each operation of the service according to our Service Ontology. An example of a catalog entry is *BLAST*, stating that *BLAST* is of type *alignment service* at URI `xml.nig.ac.jp/wsdl/Blast.wsdl`.

The *Workflow Repository* extends the layered workflow model of (Pastorello Jr. *et al.* 2005), adding information about provenance (Barga & Digiampietri 2006; 2007). Such information is stored for each workflow layer; thus, it can refer to an abstract workflow, a specific instance of a workflow, the parameters and input data of a given executable workflow or to the results of a workflow execution (including information about the produced data, steps run, etc). In other words, the Workflow Repository also stores all data needed to execute a workflow.

The *Data Transformation Rules repository* contains three kinds of information: *Default Concept Representation*, *Automatic Transformation* and *Structure Mapping*. The *Default Concept Representation* is a table that links each type in the Structural Types Repository to a concept in the Domain Ontology. That type is considered by the system as the default structural type to represent this ontological concept and it can be customized by the user. For instance, the entry `<blast_alignment,alignment>` indicates that type `blast_alignment` is the default type to represent an `alignment`. Structure Mapping and Automatic Transformation Rule are tables that contain rules to determine additional transformation across structures in a domain.

*Automatic Transformation* is a table in which each entry contains a set of rules that determine the automatic execution of workflows to convert $input\ data$ into the desired type. It is stored as $<$`domain concept, structural type, workflow id, date`$>$. The `date` indicates when the automatic transformation should be executable (for example, every night, every weekend or every time data is inserted or updated). Following the date restriction, the workflow indicated by the rule will be executed to transform the input data (if it matches the domain concept and the structural type of the rule). For example, the user can insert a rule that says that whenever a *chromatogram* is inserted in the system using the *default chromatogram structural type*, the system must execute a workflow that transforms the *chromatogram* (using a $base-calling$ activity) and produces the *DNA sequence* and the *quality* of the sequence. *Chromatograms* are binary files that contain the raw data of a *DNA sequence*.

*Structure Mapping* specifies mappings of the possible transformations among structural types in a given domain. Each entry in this table is a tuple of the form $<$`source type, target type, sub-domain, workflow id, mapping losslessness, mapping classification`$>$ where the mapping from source type to target type is executed by the workflow. The sub-domain delimits the scope in which the transformation can be applied. For example, an $integer$ (basic structural type) can always be transformed into a $float$ (another basic structural type), so the sub-domain of this transformation is our full Domain Ontology. On the other hand, a *flat file* can only be transformed into a *xml blast file* if the two structures refer to the concept

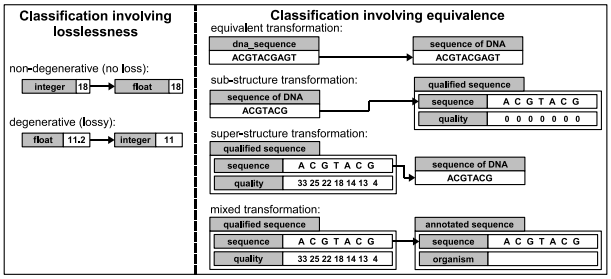


Figure 2: Data Transformation Classification

*blast alignment* from the Domain Ontology. This kind of data transformation mapping is especially useful when considering data and tools that are heterogeneous and from distinct sites.

Each transformation can be classified following two criteria: losslessness and equivalence (Santanchè & Medeiros 2005). A transformation can be *degenerative* (lossy), when there is some loss in data precision (e.g., converting float numbers to integer numbers) or *non-degenerative* (lossless), when there is no loss of information. In terms of equivalence, the results of a transformation can be *equivalent*, *super-structure* (super-type), *sub-structure* (sub-type) or *mixed*. The results are *equivalent* when all information from the source structural type is used (and sufficient) to fill all the information in the target structural type. An example is when the difference between the two types is only the name of a field – such as, if the source structure has a field called *dna_sequence* of type $string$ and the target structure has the "same" $string$ field but with the name *sequence of DNA*. The result of a transformation is a *super-structure* when the information from the source structure is enough to fill the target structure but some of the source information is not used – e.g., when transforming a structure that has a *DNA sequence* and the *quality* of this sequence into a structure that has only the *DNA sequence*. The result of a transformation is a *sub-structure* when the information from the source structure is totally used in the transformation but it is not enough to fill all the information in the target (e.g., when transforming a structure that has only a *DNA sequence* into a structure that has *DNA sequence* and the *quality* of this sequence), in this case, the missing information can be filled with $default$ or $null$ values. The results of a transformation are considered *mixed* when there is information in the source structure that is not used in the target structure and there is information in the target structure that cannot be filled by the source structure. Figure 2 illustrates these data transformation examples.

## Data Manager

The three main functionalities of the *Data Manager* Layer are:

1. It serves as the middleware (interface) between the other modules and the repositories;
2. It executes the automatic data transformations using the Data Transformation Rules;

3. It supports traceability requests.

The middleware functionality (1) provides data independence - internal storage structures can change without affecting other modules.

The execution of automatic data transformations (2) is useful to preprocess information (accelerating subsequent queries involving this information), to convert data to a desired type, and to facilitate traceability functionalities. For example, suppose a user declares that the input of a workflow in the *Workflow Repository* is a *chromatogram*; there is little information that can be extracted from this, since a chromatogram is a raw binary file. However, if an automatic transformation is executed to transform the chromatogram into a *sequence of nucleotides* with *sequence of quality values*, it is possible to pose queries on sequence alignment, search for patterns, etc.

The third functionality of the Data Manager is to provide traceability mechanisms. To start, it supports basic queries about data and metadata – e.g., "show all sequences from *LBI-UNICAMP*", or "show all *sequence alignments* produced by *blastp* alignment tool". Moreover it allows queries about the process that produced the data, as well as queries on temporal dependencies among activities – e.g. show the experiments that used the *blastn* alignment tool whose input was processed by *phred* base-calling tool.

To allow traceability processing, it also supports an extended set of domain-specific operations, for example, the *sequence alignment operation*. This allows formulation of complex traceability queries – e.g., "select all the *chromatograms* whose *DNA sequence* aligns with a given sequence".

### Interactions Between Data Manager and the Other Modules

The Data Manager provides the required functionality for data integration and traceability. It translates requests from the other modules to the data Repositories and vice-versa, providing data independence. This simplified the specification of the other modules of the architecture and ensured it extensibility. Here, we briefly describe the messages exchanged between these modules and the Data Manager. A detailed description of the modules' functionality can be found in (Digiampietri *et al.* 2006; Digiampietri, Pérez-Alcázar, & Medeiros 2007).

The *Service Register* module lets users register services into the Repository layer, via the Data Manager. The Data Manager provides information about the ontologies that are used to classify the Web service operations. The user must classify each operation offered by a service according to some type (from the Service Ontology) and must assign an ontological concept (from the Domain Ontology) to each operation parameter. Parameters structural types can be deduced from the WSDL specification of the service.

The *Service/Workflow Discovery* module searches for services or workflows that can perform a given task. It starts by requesting from the Data Manager the list of services of a given ontological type from the Service Ontology, and services with a given interface which produce results of a given concept type from the Domain Ontology. Whenever no suitable service or workflow is found, the Service/Workflow Discovery module notifies the user, who takes the appropriate action – e.g. designing a new workflow. More details can be found in (Digiampietri *et al.* 2006; Digiampietri, Pérez-Alcázar, & Medeiros 2007).

The *Design* module provides support to workflow design and composition. It asks the Data Manager information about the services and workflows available; the input and output data of a workflow already executed; and sends to Data Manager user updates to a given workflow.

The *Service request* module asks to the Data Manager information about how to invoke a given Web service, such as the provider URL, the messages that will be sent, the structural type of the service answer, etc.

Finally, the user can pose traceability and provenance queries via the interface. These queries are forwarded to the Data Manager, which processes them, and results follow the inverse path.

## Implementation Issues

The first version of the architecture has been implemented as follows. All repositories are stored in a database under the MySQL 5.0 DBMS.

Most other modules are implemented in Java. Design facilities incorporate modules to support workflow composition and edition; one of these modules was specified using SHOP2, an Artificial Intelligence planner. The Data Manager contains the core functionalities to support traceability and provenance tasks. It is also programmed in Java, and acts as an interface between MySQL and the remaining modules.

### Domain Specific Query Operators

The Data Manager encapsulates transformation mechanisms that let the other modules pose queries that have domain specific operators – e.g., *base calling*, *alignment*, etc. It translates these queries into a sequence of basic MySQL 5.0 queries, to be executed on the underlying database.

Domain specific operators can be used in two clauses: *SELECT* and *WHERE*. Let *op* be a domain specific operator and *y* some repository element to which it is meaningful to apply *op*. Thus, *SELECT op(y) FROM ...* first retrieves *y* from the repositories according to the query predicate in the *WHERE* clause, and then computes *op(y)* – executing *op* on *y*.

The same kind of procedure is applied in the *WHERE* clause: the predicate to be satisfied is the result of the computation of *op(y)*.

Let us first exemplify the *SELECT* clause. A user wants to see all the *chromatograms* that have been used/referenced in any workflow stored in the *Workflow Repository* – which, we recall, contains workflows at several abstraction levels, data, provenance information, and annotations. However, since chromatograms are stored in binary files, the user wants to see only the *id* of the chromatograms and their *nucleotide sequence*. The *nucleotide sequence* can be produced by the *base_calling* domain specific operator. However, this operator returns objects of type *sequence_and_quality*, which

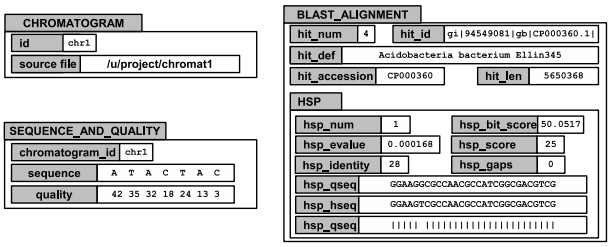


Figure 3: Examples of objects of types *chromatogram*, *sequence_and_quality* and *blast_alignment*

have three attributes – see example of such an object in Figure 3. In this case, the user must specify which components of *sequence_and_quality* objects are to be returned, using the standard *dot* notation of query languages. The select statement to execute this query is:

```
SELECT chromatogram.id,
       base_calling(chromatogram).nucleotide_sequence
       FROM WorkflowRepository
```

Consider now a *WHERE* with special computations. For example, suppose a user wants to retrieve all *reads* that align with a given *read* whose *identity* is greater than or equal to a given value:

```
SELECT read FROM WorkflowRepository WHERE
    alignment(read,'GGAAGGCGCCAACGCCATCGGCGACGTCG').hsp_identity >= 25
```

A *read* is the sequence of letters $A$, $C$, $G$, and $T$ extracted from a chromatogram; *alignment* is a domain specific operator whose result is represented in the form of a *blast_alignment* structure (see Figure 3). This operator receives as input two *DNA sequences* (*read* is a sub–type of *DNA sequence*) and produces an object of the type *blast_alignment*. This operator invokes a tool called BLAST (Basic Local Alignment Search Tool) (Altschul *et al.* 1997) to execute this operator. Blast_alignment is a structure composed by several fields (based in the XML blast model (Madden 2002); we highlight the fields named hit_num (the number of the alignment), the hsp_score (the score of the alignment) and the hsp_identity value (the number of nucleotides that matched in the alignment).

## Tool and Data Traceability

The implementation of traceability mechanisms depends on two factors: (i) a storage infrastructure that stores all relevant information about the data and the processes; and (ii) tools to support forward and backward traceability.

Section Repositories presented our storage infrastructure. The second factor depends on the user's needs. This section presents an example of a sequence assembly experiment, for which we discuss six traceability queries to illustrate specific points.

Suppose a user wants to assemble a set of DNA sequences to produce the *assembly_result*. This set is composed of three subsets: `read1` (reads extracted from chromatograms and inserted in the system by the user); `read2` (reads inserted by the user but without corresponding chromatograms) and `contigs` (preassembled reads). After being assembled, not all of the sequences may be present in the

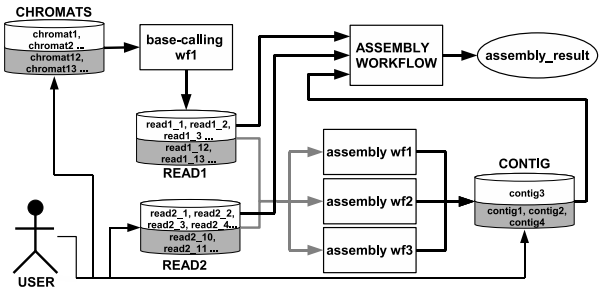


Figure 4: Example of an assembly experiment.

*assembly_result* (which always happens in big assemblies). The input sequences subsets can be redistributed in six groups: `read1in`, `read1out`, `read2in`, `read2out`, `contigsIn` and `contigsOut` – where the suffixes mean that the group of sequences is present in the *assembly_result* (`in`) or not (`out`). Note that it is only possible to know if a sequence (or parts thereof) was actually used in the assembly experiment after the execution of the assembly workflow, and then only if the assembly workflow stores (as part of its result) a description of the read composition of the resulting assembled sequence.

Figure 4 shows a graphical representation of this experiment. READ1, READ2, and CONTIG correspond to the subsets `read1`, `read2`, and `contigs`, respectively. CHROMATS corresponds to the raw data from the chromatograms (before the *base–calling* operation, invoked to extract chromatogram fields sequence and quality – see Fig. 3). The data in the datasets CHROMATS, READ1 and CONTIG was partially inserted by the users and partially produced by the system. To simplify the explanation, let us consider that the CONTIG dataset has only four contigs (called `contig1`, `contig2`, `contig3` and `contig4`). The rectangles in the figure correspond to workflows. There are five workflows: *base–calling wf1*, which converts data from CHROMATS to READ1 *assembly_wf1*, *assembly_wf2*, *assembly_wf3*, which produce, respectively, `contig1`, `contig2` and `contig3`; `contig4` was inserted by the user. *ASSEMBLY WORKFLOW*, the fifth workflow, produces the *assembly_result*. The elements in gray are the sequences that are present in the *assembly_result* – that is, they belong to the groups `read1in`, `read2in` and `contigsIn`.

When querying the databases, the user can choose between two kinds of answers: nonrecursive (which shows only the immediate result of a query) and recursive (which recursively navigates in an experiment to give a more complete traceability answer). The latter is useful in the case of a workflow with composite activities (i.e., where one activity itself encapsulates a workflow). This supports traceability at distinct abstraction levels. There follows a six representative traceability demands, two of which (1 and 5) involve data used in processes, and the others process execution details.

1. **What is the input data that was used to run the ASSEMBLY WORKFLOW?** The nonrecursive answer is

the data from the sets `read1`, `read2`, and `contigs`. The recursive answer also includes the data from CHROMATS that generates the data from the set `read1`, and all the DNA sequences and chromatograms that were used to produce the contigs (from the `contigs` set). This question requires backward tracing and its answer involves all data used in a workflow execution. It is useful when a user intends to re-execute an experiment and needs all the input information to do this.

2. **What are the input sequences that are present in the *assembly_result*?** The answer is equivalent to the previous one, but will contain sequences from `read1in`, `read2in`, and `contigsIn`. This question also requires backward tracing and is useful to distinguish the input data (query 1) from the data actually used.
3. **What processes were used to produce *assembly_result*?** The nonrecursive answer is only *ASSEMBLY WORKFLOW*. The recursive answer also includes *assembly wf1*, *assembly wf2* and *base–calling wf1*. The answer - a set of processes (workflows) - uses backward tracing. Note that `contig3`, the data produced by *assembly wf3*, was not used in the *assembly_result*).
4. **What services/tools were used to produce *assembly_result*?** This requests all the services that compose the workflows from the previous query. It uses backward tracing that, recursively, looks inside each workflow for services used to produce the *assembly_result*.
5. **What data was produced using raw chromatogram number 1 (from the CHROMATS dataset)?** The result is the DNA sequence produced through the use of the *base–calling* workflow and all the assemblies that use this sequence. This needs forward tracing and is useful when a user wants to update a given information and needs to know what data derives from this information.
6. **What services or processes produce an *alignment*?** The answer is the set of all services and workflows whose outputs produce an *alignment* or any sub–concept (in the Domain Ontology) of *alignment*. This requires a combination of interface matching and semantic relationships, being useful when a user wants to know what tools can produce a desired concept (or data type).

We point out that we provide traceability functionality that is normally unavailable in standard mechanisms. This is achieved thanks to the fact that we store extended semantic knowledge about each tool and process used in an experiment.

For instance, query 2 wants to know which of the input sequences effectively contributed to construct the result. This cannot be deducted in standard representations, where the only information available is some kind of tuple $<$`input data, process, output`$>$. Our repository management mechanisms instead, annotate processes and tools with a sophisticated type facility. Here, for instance, our system knows that this execution of *Assembly Workflow* invoked a specific tool whose results include logs of its intermediate steps. The processing of query 2 is thus transformed into a set of queries that first identify the type of the tool used in the assembly and, afterward, process all of its logs, themselves annotated with structural types and ontology terms.

## Conclusions

This paper presented a framework to support bioinformatics experiments enhanced with traceability mechanisms. This framework helps scientists to record detailed information about all processes involved in an experiment. Our solution takes advantage of structure mapping to facilitate data integration, and of interface matching to provide interoperability. It also extends database query facilities to support domain-specific concerns. The framework has been validated by a prototypical implementation with real data.

Several bioinformatics laboratories use a scientific workflow infrastructure to design and execute their experiments - e.g., (Hoon *et al.* 2004; Oinn *et al.* 2004; Stevens *et al.* 2004). Our work extends these approaches through the use of the Data Manager layer that, when combined with data repositories, provides interoperability, data integration and traceability mechanisms.

As future work we intend to specify and implement a ranking algorithm to sort the results of a query, based on the suitability of each result. We also intend to develop mechanisms for sharing transformation rules. Finally, we intend to develop graphical tools to facilitate data presentation and include user feedback loops to improve retrieval facility.

## Acknowledgments

The work described in this paper was partially financed by Brazilian funding agencies: FAPESP, CAPES and CNPq, and by a Microsoft Research Fellowship.

## References

Altschul, S.; Madden, T.; Schaffer, A.; Zhang, J.; Zhang, Z.; Miller, W.; and Lipman, D. 1997. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.* 25:3389–3402.

Barga, R., and Digiampietri, L. 2006. Redux - First provenance challenge. http://twiki.ipaw.info/bin/view/Challenge/REDUX (as of 2007-04-04).

Barga, R., and Digiampietri, L. 2007. Automatic capture and efficient storage of escience experiment provenance. *Accepted for publication in Concurrency and Computation:Practice and Experience*.

Bowers, S., and Ludäscher, B. 2004. An ontology-driven framework for data transformation in scientific workflows. In Rahm, E., ed., *Proceedings of DILS*, volume 2994 of *Lecture Notes in Computer Science*, 1–16. Springer.

Buttler, D.; Coleman, M.; Critchlow, T.; Fileto, R.; Han, W.; Pu, C.; Rocco, D.; and Xiong, L. 2002. Querying multiple bioinformatics information sources: can semantic web research help? *ACM SIGMOD Record* 31:56–64.

Cambridge Dictionaries Online. http://dictionary.cambridge.org (as of 2006-12-18).

Cavalcanti, M.; Targino, R.; Baiao, F.; Rössle, S.; Bisch, P.; Pires, P.; Campos, M.; and Mattoso, M. 2005. Managing structural genomic workflows using Web services. *Data & Knowledge Engineering* 53(1):45–74.

Diehn, M.; Sherlock, G.; Binkley, G.; Jin, H.; Matese, J.; Hernandez-Boussard, T.; Rees, C.; Cherry, J.; Botstein, D.; Brown, P.; and Alizadeh, A. 2003. SOURCE: a unified genomic resource of functional annotations, ontologies, and gene expression data. *Nucleic Acids Research* 31(1):219–223.

Digiampietri, L.; Pérez-Alcazar, J.; Medeiros, C.; and Setubal, J. 2006. A framework based on semantic Web services and AI planning for the management of bioinformatics scientific workflows. Technical Report IC-06-04, Institute of Computing, University of Campinas.

Digiampietri, L.; Pérez-Alcázar, J.; and Medeiros, C. 2007. An ontology-based framework for bioinformatics workflows. *Accepted for publication in Int. J. Bioinformatics Research and Applications*.

Hernandez, T., and Kambhampati, S. 2004. Integration of biological sources: Current systems and challenges ahead. *SIGMOD Record* 33(3):51–60.

Hollingsworth, D. 1995. The Workflow Reference Model. Technical Report TC-1003, Workflow Management Coalition.

Hoon, S.; Ratnapu, K.; Chia, J.; Kumarasamy, B.; Juguang, X.; Clamp, M.; Stabenau, A.; Potter, S.; Clarke, L.; and Stupka, E. 2004. Biopipe: A flexible framework for protocol-based bioinformatics analysis. *Genome Research*.

Madden, T. 2002. The blast sequence analysis tool. http://www.ncbi.nlm.nih.gov/books/bookres.fcgi/handbook/ch16d1.pdf (as of 2007-01-19).

Maletic, J.; Collard, M.; and Simoes, B. 2005. An XML based approach to support the evolution of model-to-model traceability links. In *TEFSE '05: Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*, 67–72. New York, NY, USA: ACM Press.

Medeiros, C.; Perez-Alcazar, J.; Digiampietri, L.; Pastorello, G.; Santanche, A.; Torres, R.; Madeira, E.; and Bacarin, E. 2005. WOODSS and the Web: Annotating and Reusing Scientific Workflows. *ACM SIGMOD Record* 34(3):18–23.

Moreau, L., and Foster, I., eds. 2006. *Provenance and Annotation of Data, International Provenance and Annotation Workshop, IPAW 2006, Chicago, IL, USA, May 3-5, 2006, Revised Selected Papers*, volume 4145 of *Lecture Notes in Computer Science*. Springer.

Oinn, T.; Addis, M.; Ferris, J.; Marvin, D.; Senger, M.; Greenwood, M.; Carver, T.; Glover, K.; Pocock, M.; Wipat, A.; and Li, P. 2004. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics* 20:3045–3054.

Opara, L. 2003. Traceability in agriculture and food supply chain: a review of basic concepts, technological implications, and future prospects. *Food, Agriculture & Environment* 1(1):123–125.

Pastorello Jr., G.; Medeiros, C.; Resende, S.; and Rocha, H. 2005. Interoperability for GIS Document Management in Environmental Planning. *Journal of Data Semantics* 3(LNCS 3534):100–124.

Peterson, J.; Umayam, L.; Dickinson, T.; Hickey, E.; and White, O. 2001. The comprehensive microbial resource. *Nucleic Acids Research* 29(1):123–125.

Santanchè, A., and Medeiros, C. 2005. Self describing components: Searching for digital artifacts on the web. In *Proceedings of Brazilian Symposium on Databases (SBBD)*.

Setubal, J., and Meidanis, J. 1997. *Introduction to Computational Molecular Biology*. Boston: PWS Publishing Company.

Stein, L.; Mungall, C.; Shu, S.; Caudy, M.; Mangone, M.; Day, A.; Nickerson, E.; Stajich, J.; Harris, T.; Arva, A.; and Lewis, S. 2002. The generic genome browser: A building block for a model organism system database. *Genome Research* 12:1599–1610.

Steinauer, D.; Wakid, S.; and Rasberry, S. 1997. Trust and traceability in electronic commerce. In *StandardView*, volume 5, 118–124.

Stevens, R.; Tipney, H.; Wroe, C.; Oinn, T.; Senger, M.; Lord, P.; Goble, C. A.; Brass, A.; and Tassabehji, M. 2004. Exploring Williams-Beuren syndrome using myGrid. *Bioinformatics* 20(1):i303–310.

Wainer, J.; Weske, M.; Vossen, G.; and Medeiros, C. 1996. Scientific Workflow Systems. In *Proc. of the NSF Workshop on Workflow and Process Automation Information Systems*.