An ontology-based framework for bioinformatics workflows

Luciano A. Digiampietri*

Institute of Computing,
University of Campinas,
CP 6176, 13084-971, Campinas, SP, Brazil
E-mail: luciano@ic.unicamp.br
*Corresponding author

José de J. Pérez-Alcázar

EACH, University of São Paulo,
São Paulo, SP, Brazil
E-mail: jperez@usp.br

Claudia Bauzer Medeiros

Institute of Computing,
University of Campinas,
CP 6176, 13084-971, Campinas, SP, Brazil
E-mail: cmbm@ic.unicamp.br

Abstract: The proliferation of bioinformatics activities brings new challenges – how to understand and organise these resources, how to exchange and reuse successful experimental procedures, and to provide interoperability among data and tools. This paper describes an effort toward these directions. It is based on combining research on ontology management, AI and scientific workflows to design, reuse and annotate bioinformatics experiments. The resulting framework supports automatic or interactive composition of tasks based on AI planning techniques and takes advantage of ontologies to support the specification and annotation of bioinformatics workflows. We validate our proposal with a prototype running on real data.

Keywords: bioinformatics workflows; bioinformatics ontologies; bioinformatics tool composition; bioinformatics data and tool annotation.

Reference to this paper should be made as follows: Digiampietri, L.A., Pérez-Alcázar, J.J. and Medeiros, C.B. (2007) 'An ontology-based framework for bioinformatics workflows', *Int. J. Bioinformatics Research and Applications*, Vol. 3, No. 3, pp.268–285.

Biographical notes: Luciano Antonio Digiampietri is a PhD student at the Institute of Computing, University of Campinas (UNICAMP), Brazil. He has been awarded a Microsoft Research Fellowship. His research interests are databases, information systems and bioinformatics.

José de Jesus Pérez-Alcázar is a Lecturer at the School of Arts, Science and Humanities of the University of São Paulo, Brazil. He received his PhD from the Catholic University of Rio de Janeiro (PUC-Rio). His research interests are AI planning, semantic web and workflow management systems.

Claudia Bauzer Medeiros received his PhD from the University of Waterloo in 1985 and is full Professor of Computer Science at the Institute of Computing, UNICAMP, Brazil. Her projects centre on design and development of scientific database applications, with emphasis on geographic data, agro-environmental planning, bioinformatics and biodiversity. She has been (co) Principal investigator of several multi-disciplinary and multi-national projects, involving partners in Germany, France, Argentina, Chile and the USA, and holds a visiting scholar cooperation with the Dauphine University, France. She serves on the editorial boards of the *VLDB Journal and GeoInformatica* and is a member of ACM and IEEE and President of the Brazilian Computer Society (2004–2007).

1 Introduction

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). They differ from business workflows in several points. In particular, in bioinformatics they are characterised by a high degree of human intervention and variability in workflow design for the same task.

Bioinformatics workflows are often specified manually and tasks are redefined from scratch (e.g., using script languages (Cavalcanti et al., 2005)). With the advent of distributed execution of workflows (e.g., in grids (Stevens et al., 2004)), task definition is sometimes being replaced by the invocation of a web service that performs that task (Gao et al., 2005).

Manual composition is hard work and susceptible to errors. In bioinformatics, due to the constant evolution of the area and the combinatorial explosion of tools, there are too many alternatives for workflow construction and choice of appropriate services. Thus, there is a need for means to help scientists to design appropriate workflows. Another important issue is that of traceability, to ensure the quality of an experiment.

The main idea behind our work is to take advantage of ontologies to support the specification and annotation of bioinformatics workflows and to serve as the basis for tracking data provenance (Fileto et al., 2003). An underlying assumption is that the problem of automatic or iterative composition of workflow tasks can be seen as an AI planning problem (Long and Fox, 2003). We extend AI planning techniques with ontologies to create a semantic framework for design, reuse and annotation of bioinformatics experiments.

The paper attacks the problem of constructing and annotating scientific workflows, under the assumption that they are the basis for specifying and executing tasks in a distributed (laboratory) environment. Each activity within such a workflow can be executed either by invocation of a web service or of another (sub) workflow.

The paper's main contributions are thus:

- proposing a solution to the problem of composition of services, combining results from AI and ontology management, thereby helping design scientific workflows, while at the same time documenting design alternatives
- using ontology repositories to enhance the semantics in automatic workflow construction and facilitate tracking data and procedure provenance
- validating the proposal by means of a prototype for genome assembly and annotation.

Our implementation takes advantage of WOrk-flOw-based spatial Decision Support System (WOODSS) (Medeiros et al., 2005), a scientific workflow infrastructure. Originally conceived for decision support in environmental planning, it has evolved into an extensible database-centred environment that supports specification, reuse and annotation of scientific workflows and their components.

The rest of the paper is organised as follows. Section 2 describes related work. Section 3 presents our architecture. Section 4 discusses our prototype. Section 5 contains conclusions and ongoing work.

2 Related work and concepts

2.1 Workflows and web services in bioinformatics

The *genome assembly* problem consists in joining and matching together pieces of DNA sequences to create a cogent sequence, much in the way crossword puzzles are put together. Constituent sequences are created inside a laboratory by procedures that extract pieces from a species' DNA and then produce long strings of base pairs (ACGT). Challenges in this process include the adequate generation and annotation of sequences, as well as finding the appropriate means of assembling them together into an accepted genome.

Genome annotation is the assignment of functions to each gene. The empiric verification of gene functions is a time and money consuming activity. Most functions are assigned based on the similarity between the DNA sequence of the target gene and the sequences of already annotated genes. Gene annotation can therefore be partially automated, but manual data verification is always recommended.

Genome assembly and annotation are composed of several complex activities involving interactions among basic tasks, human intervention and access to heterogeneous data sources. A trend in the bioinformatics community is to see each such experiment as a workflow designed by scientists to help their daily activities (Oinn et al., 2004). However, this practice has little flexibility, hampering the edition and reuse of these workflows.

Several other problems are involved in the construction of such workflows. Among them, this paper is concerned with:

- data and software tool provenance
- tool/task composition, translated into a problem of web service composition
- mechanisms for finding the appropriate tools or services to execute a task.

We use domain ontologies as the basis for attacking these problems. The first question – provenance of data and software tools – directly affects the acceptance of the results of experiments. The quality of bioinformatics experiments depends on properly identifying data origins and the processes that produced these data (Buttler et al., 2002). At most times, provenance is indicated by laborious manual annotations, which often vary across laboratories.

The second issue concerns tool/task composition while constructing the workflows (Cavalcanti et al., 2005; Yu and Buyya, 2005; Medeiros et al., 2005). We highlight three kinds of composition: manual (supervised), iterative (using top-down design practices) and automatic. In a scenario where several software tools are being made available on the web, the composition problem has become more important. To help this issue, many tools invoked by such workflows are now encapsulated into web services.

Our third problem is to find tools on the web that execute some task. This search is typically based on keyword queries (Buttler et al., 2002) that can, however, return several unwanted results and may not find the desired tools. Even when found, the integration of a tool with the user's system is not easy. Thus, laboratories rebuild tools, replicating work and decreasing tool sharing and reuse.

The Semantic Web has been proposed to solve interoperability and discovery problems. However, this will require extending the languages to add semantics to service description and discovery – e.g., using ontologies. The development of Semantic Web services must address the challenges of automatic discovery, invocation, composition and monitoring of service execution.

2.2 Planning and composition of web services

Automatic composition of web services is a recent trend to meet some of the challenges and problems mentioned in the previous section. Users (in this paper, bioinformatics scientists) should be able to specify 'what' they desire from the composition (high level goals and actions) and the system supplies the 'how' – the web services to be used, how to interact with those services, etc. The process of composing the services must be transparent to the users and the detailed descriptions of the composed services must be generated automatically by the system from the users' specifications.

Among the proposed solutions for the automatic composition problem we mention those based on planning, exploited by us and those based on workflows. Workflow-based composition methods can be divided into static and dynamic workflow generation (Rao and Su, 2004). In the first case, the workflow is specified manually and only the selection and binding of atomic web services with the workflow is automatic. Dynamic composition automatically creates the workflow and selects atomic services, e.g., da Costa et al. (2004) and Fujii and Suda (2004).

The task of presenting a sequence of actions to achieve an objective is called in AI *plan synthesis*, or *planning* (Ghallab et al., 2004; Russel and Norvig, 2003). Planning is a mature area in AI, with well-studied algorithms. Such techniques are currently used in mobile robots, manufacturing processes and emergency management, among others (Munoz-Avila et al., 2001; Nau et al., 1995).

Recent research efforts have investigated the use of planning to solve the problem of automatic composition of web services (Blythe and Ambite, 2004). According to Srivastava and Koehler (2003), in order to use planning in this context, AI planning

concepts must be extended to consider factors such as complex control structures with loops, non-determinism and conditionals.

We highlight other important characteristics, not usually found in AI planning, such as: the use of non-functional attributes, like cost or quality, which can facilitate the choice of the plan most adequate to the user's requirements; the need to support semantic constructions such as hierarchies (abstractions), as well as compatibility with the different Semantic Web service description standards, like OWL-S (www.daml.org/services) and WSMO (www.wsmo.org); and the support of extended goals involving complex conditions on process behaviour.

Many planning systems and algorithms have been considered as candidates for supporting composition of web services. For instance, the work of (McIlraith and Son, 2002) extends Golog (Levesque et al., 1997), a language based on situation calculus, to allow automatic building of web services. Another option is the use of Planning Domain Definition Language (PDDL), a widely used formal language, whose notation is similar to OWL-S and is thus a good candidate to use in specifying web services composition. Other solutions involve rule based planning methods (Medjahed et al., 2003) and symbolic model checking, which has also been used to automatically compose services described in OWL-S (Traverso and Pistore, 2004).

Yet another method is *hierarchical planning*, to support iterative and automatic composition of services to specify workflows. Hierarchical planning is an AI planning methodology that creates plans by task decomposition. One well-known hierarchical planner is Simple Hierarchical Ordered Planner 2 (SHOP2) (Nau et al., 2003) which is based on Hierarchical Task Network (HTN) (Russel and Norvig, 2003). SHOP2 won the prize of one of the four best planners in the 2002 International Planning Competition (Long and Fox, 2003).

Sirin et al. (2004) use SHOP2 for the automatic composition of web services and the inputs to their planner are specified in OWL-S. The authors claim that automatic task decomposition using HTN planning is very similar to the concept of complex process decomposition used in OWL-S ontologies.

None of the planning proposals mentioned treats complex objects or objects created dynamically, two very important characteristics within web services. Moreover, planning algorithms do not consider the existence of relationships among objects which might result in plan improvement. As will be seen, we solve these issues by extending SHOP2 to take advantage of ontology repositories.

3 An architecture for automatic composition via planning

This section presents our solution to the problem of helping scientists design scientific workflows. As will be seen, one important issue is the possibility of reusing parts of workflows constructed by other scientists. Another relevant issue is annotation. Reuse and annotation are supported by ontologies, which also guide the planning algorithm.

Our plans are specifications of scientific workflows. Thus, in this section, we will use, indistinctly, both terms.

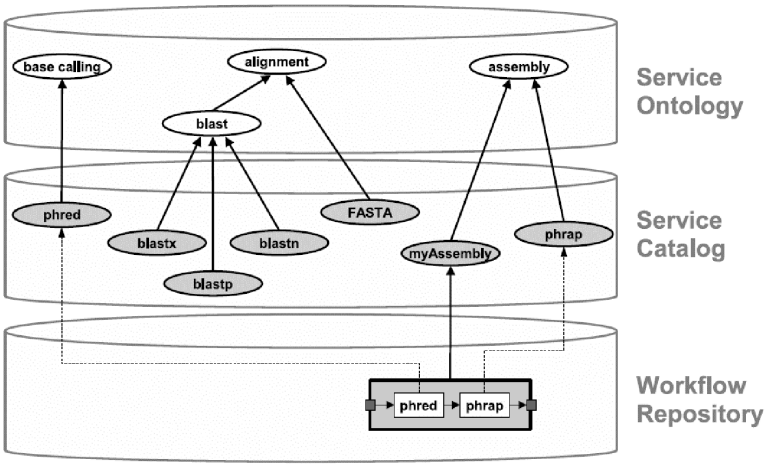
3.1 Repositories

Our workflow design and execution process is based on combining AI planning techniques with information stored in three repositories: *Ontology Repository*, *Service Catalog* and *Workflow Repository*. While the Ontology Repository contains information about domains and service types, the Service Catalog stores information about service instances.

In more detail, the *Ontology Repository* contains two kinds of ontologies (Domain and Service) that are used to support automatic composition and annotation of services and workflows – in our case study, information about genome assembly and annotation, see Section 4. The concepts in the Domain Ontology describe a given application domain. The concepts in the Service Ontology describe the different kinds of services and their relationships. This ontology is used in automatic composition to help check compatibility among composed tasks (e.g., interface matching). The Service Ontology does not store descriptions of the services themselves. Rather, it contains what we choose to call 'service type' – i.e., a generic description of each kind of service, its generic interface, parameters, etc. Thus, it will contain a description of 'alignment' services, but no instantiation of this type of service. Service instantiation is left to the Service Catalog.

Figure 1 shows the 'is a' relations of some services in the Service Ontology (service types) and the services in the Service Catalog (service instances) – e.g., a FASTA service 'is a' service of the type alignment. Our repositories also store 'aggregation' relations, for example, the service *myAssembly*, in the Service Catalog, can be associated with a workflow, in the Workflow Repository, that is an aggregation of the services *phred* and *phrap*. This kind of relationship is used in our iterative composition process (in each iteration, the system suggests to the user how to decompose an abstract activity into more concrete activities).

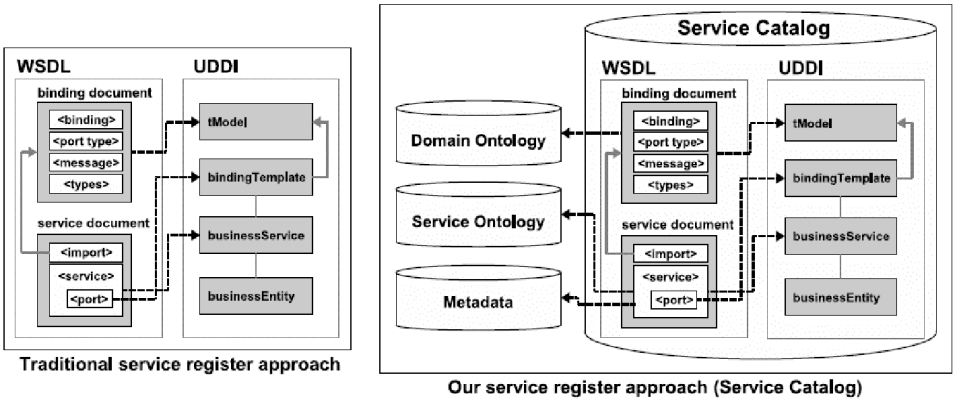
Figure 1 Parts of our service ontology. Service types appear in white ovals, service instances in dark ovals



The *Service Catalog* plays the role of a Universal Description Discovery and Integration (UDDI) registry, enhanced with extended functionalities. Standard UDDI structures

store information about service providers, the web services they make available and the technical interfaces which may be used to access those services ports. Our Service Catalog, besides, stores a service's non-functional attributes (metadata), such as execution time, quality, reliability and availability, used to rank workflows. Figure 2 presents the standard UDDI web services register approach and our extension to this approach. Each service entry in the Catalog is annotated with the ontological concepts of the Service Ontology. The port types are annotated with concepts from the Domain Ontology.

Figure 2 Service register approach: UDDI approach and our semantic approach



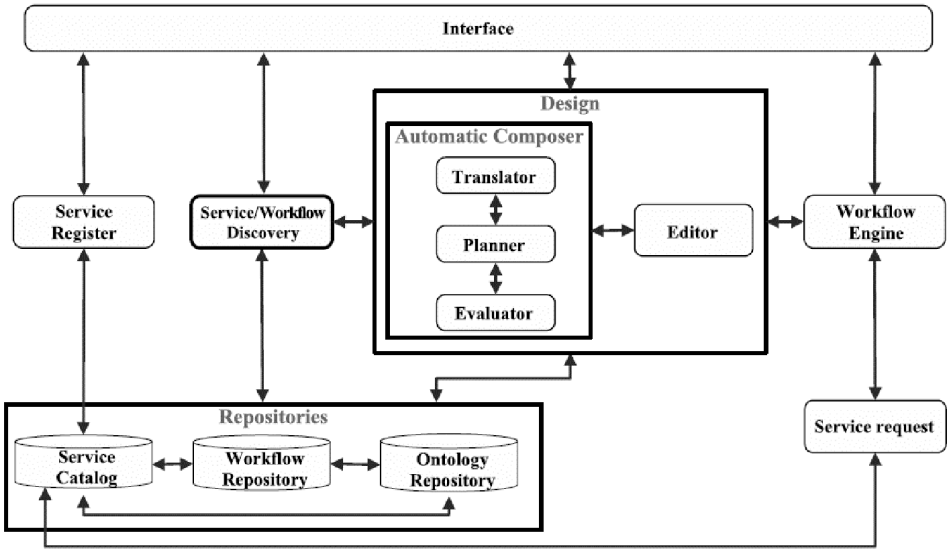
The *Workflow Repository*, adopted from WOODSS (Medeiros et al., 2005), stores annotated (sub) workflows at different abstraction levels, from abstract specifications to runtime workflows. The Workflow Repository also stores all annotations and information on data needed to run a given executable workflow. This includes pointers to files that store intermediate execution results and metadata associated with each execution (e.g., timestamps, actors involved).

The repositories are interrelated as follows. Workflows and data in the Workflow Repository are annotated with terms from both ontologies; the services invoked within concrete workflows come from the Service Catalog. Moreover, concepts of the Service Ontology are used to annotate terms in the Service Catalog; and concepts in the Domain Ontology annotate types stored in the Service Ontology. Section 4.1 discusses these interrelationships for our case study.

3.2 The composition architecture

Our architecture is able to deal with automatic composition of workflows based in web services. Figure 3 shows this architecture, highlighting the main modules and their interactions. Only the main connections among modules are represented. Section 4 presents our implementation of this architecture.

Figure 3 System architecture



The Interface Layer allows the user to design, search, edit and execute a workflow and store it in the repository. It also allows a user to register services and workflows, request the execution of a workflow and interact with this execution. Furthermore, it supports syntax verification and suggestions of activities; automatic specification through AI planning; and iterative composition.

The Service/Workflow Discovery module is responsible for the search of services and workflows that meet user requests. Search can be based on context, syntax and functionality. Search for context is based on ontological annotations of services. Search on syntactic compatibility is based on the parameters of the services' interfaces. Search for functionality is based on keywords and can be local (Repositories) and global (on the web). Whenever the global search returns a service that is not already registered in the Service Catalog, the user is required to validate and register the appropriate answers in the local Repository. Hence, the Repository contains only properly annotated services, whose provenance and quality are guaranteed by some expert. When no service or workflow meets the requests, this module will ask the Design module to create new workflows.

The Design module is responsible for constructing a workflow, which at any time can be edited by a scientist (the Editor box). The Automatic Composer encapsulates the Translator-Planner-Evaluator modules of AI planning, see Rao and Su (2004). It receives a plan request *R* from the interface and generates workflows automatically or iteratively. To generate these workflows, the Translator needs first to convert the request to the planner's language. Next, the Planner interacts with the Workflow and the Ontology Repositories to obtain information for plan generation and with the Service/Workflow Discovery facility to check for existing available services.

Rather than generating executable workflows, our planner produces abstract workflow specifications. The reason is that plans refer to *service types* (defined in the Ontology Repository) rather than to the services themselves (whose specification is stored in the Service Catalog). This choice was made mainly to improve efficiency and

scalability in the planner (Agarwal et al., 2005). The Evaluator converts these abstract workflows into concrete ones and chooses from among them the workflow that best suits the request *R*. This selection is based on the non-functional attributes (execution time, quality, reliability, etc.) that annotate services in the Catalog and can be guided by the user. The workflow can then be forwarded to the Workflow Engine, where the user can provide the input data and start its execution.

The Editor module supports workflow design. It accesses the workflow repository and lets the user manually compose, reuse and annotate workflows. Annotations include free text and references to the ontology repository. This repository can also be updated (e.g., adding or modifying an ontology). However, this is outside the scope of the architecture and is left to specialised tools – e.g., Protégé (Knublauch et al., 2004) – since the architecture's goal is not to manipulate ontologies but to use them to help experiment annotation, reuse and specification.

The user interacts with the Service Register module in order to define new services. These services are described in WSDL and OWL-S and linked to the Ontology Repository. These services can be those developed and available locally, or those that are available elsewhere, but whose provenance has been certified by the user. New service types are registered in the Ontology Repository using an ontology editing tool. New services are entered into the Service Catalog using the Service Register Module.

3.3 Execution environment

The Workflow Engine follows the specification of the Workflow Management Coalition (http://www.wfmc.org). It is responsible for workflow execution and supports user interaction, e.g., to validate or interrupt execution flow. It is responsible for controlling the execution of all workflow activities. The operations provided by the Workflow Engine are: interpretation of the complex process definitions; creation and management executable workflows; and supervisory and management functions. It sends the requests (and parameters) for service invocation to Service Request.

The Service Request module is responsible for the management of each web service request, communicating with the web server provider, sending input data and receiving the results. This module also detects service faults. Faults are registered and used to calculate service non-functional attributes (such as availability).

There are three alternatives to solve a fault. The first is to try and re-execute the service that presented the fault. This is useful when a service is unavailable during a short period of time. The second is to replace the faulty service by an equivalent service (of the same type but with less ranking, following non-functional attributes, in our Service Catalog). In this case, the Workflow Engine module can ask the Evaluator module for the selection of an alternative equivalent service to replace the faulty service. If these alternatives do not work, the Workflow Engine can request new plans to solve the problem.

This architecture supports the three kinds of composition presented in Section 2.1. In manual composition, the system will only let a user combine two activities if their inputs and outputs are ontologically compatible. Ontological compatibility is based on subsumption properties, see Fileto (2003). In iterative composition, in each iteration the system suggests to the user tasks or sub-workflows that have been previously stored in the repository and that can be used for an already defined task. In automatic composition, it designs a set of workflows that satisfy the user's requests.

3.4 User interaction

Users can interact with the architecture to annotate and design workflows, monitor and change their execution and register services. One of the most important user interactions is the request for a plan (i.e., the construction of a new workflow to meet a specific request). This process starts when a user (human or software agent) makes a request for a service.

This request can be the description of a goal or task. Starting from here, the Planner generates alternative plans to meet the request. In this process, the planner accesses the Domain and Service ontologies to obtain the necessary information for the planning process. Once the plans are generated, they are passed on to the Evaluator, which chooses the best plan to meet user needs.

Domain and Service ontologies, stored in the ontology bases, are key concepts to this process. Initially, they are used by the Translator to generate a request to the planner, disambiguating the user's demand for a service. Next, these bases are used by the Planner to generate the appropriate service compositions. The Planner accesses them to obtain the functionalities of the services and generates an abstract scientific workflow. The Planner uses the domain ontology to improve the efficiency of the planning process and to facilitate the modelling and the management of complex objects. The planner's output contains several workflows (the plans) with equivalent or similar functionalities.

4 Case study: genome assembly and annotation

We implemented a prototype of the architecture presented in Section 3 to solve the problems of genome assembly and annotation. We decided to use SHOP2 in our implementation because it provides the following benefits:

- it supports embedding domain knowledge to control the search space and improve efficiency
- it has been successfully used in a variety of real-world planning-based applications
- it allows inclusion of different types of pre-condition constraints for service operators as well as calls to external systems
- it enables reuse by facilitating selection of appropriate methods from domain-related operator libraries.

Section 4.2 shows how we extended the SHOP2 language to support complex objects and enhanced semantics.

4.1 Repository instantiation

In order to implement the architecture, we had to construct the appropriate ontologies. Ontology editing uses Protégé (Knublauch et al., 2004), a well known ontology editing tool.

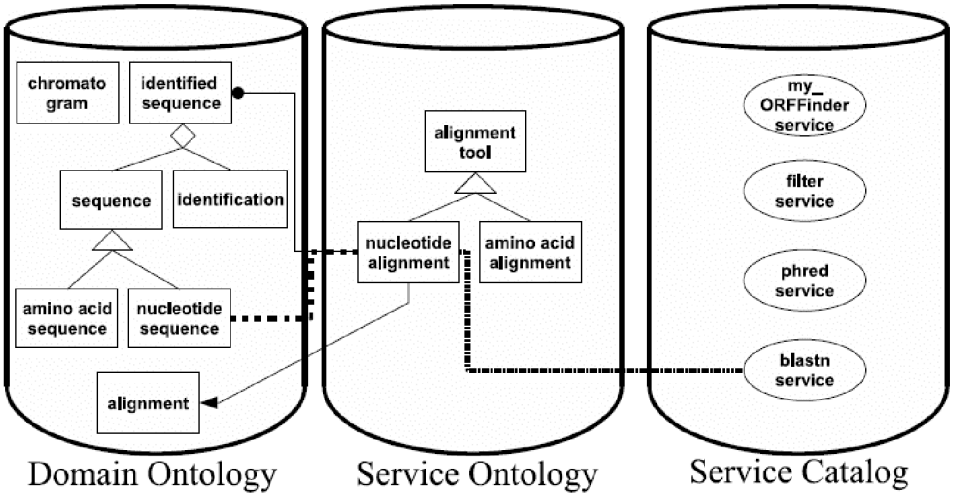
Several bioinformatics ontologies have been proposed (Smith and Schulze-Kremer, 2003; Stevens et al., 2000). Some are very detailed taxonomic ontologies, used to cluster (Yoo and Hu, 2006) and/or to annotate data (Smith and Schulze-Kremer, 2003).

Others are concerned with describing relationships among bioinformatics tools (akin to our service ontology) or objects manipulated in bioinformatics (e.g., our domain ontology). The latter are still under development and are usually defined independently. This hampers their use in an integrated manner. Ontology management techniques, such as alignment or integration (Kalfoglou and Schorlemmer, 2003) are not suitable for this kind of need, since they apply when ontologies cover associated concepts.

Since our goal was to help scientists to specify and manage their experiments, we developed detailed domain and service ontologies, specific to genome assembly and annotation, extending a generic bioinformatics ontology (Stevens et al., 2000) and specifying relationships between these two ontologies. As a consequence, they can be considered to form a (single) complex ontology covering services and domain aspects.

Using our ontology repository, we annotated bioinformatics data and tools in order to allow semantic search and automatic composition of services. Figure 4 shows a small portion of our ontologies and their interrelationships. Domain and Service portions are separated, thus helping establish distinct relationships among the concepts. In particular, the figure shows how concepts in the Service Ontology help qualify services and how concepts in the Domain Ontology help define service parameters. To simplify the figure, we omitted several relationships.

Figure 4 Small example of the relationships among our repositories



We highlight only the relationships (links among the repositories) involved with *nucleotide alignment*. The *blastn service* entry in the Service Catalog corresponds to a service that implements a *nucleotide alignment*, which is an *alignment tool* description (Domain Ontology annotation). The *nucleotide alignment* has as input an *identified sequence* and as output an *alignment*, both concepts of the application domain, duly annotated by appropriate references in the ontology. The dotted line between *nucleotide alignment* and *nucleotide sequence* indicates a restriction on input data: the *identified sequence* input to the *blastn* service must be a *nucleotide sequence*.

In our Domain Ontology, all concepts are atomic data types (integer, strings, etc.) or, recursively, an aggregation and/or a generalisation (or specialisation) of concepts. We can observe in Figure 4 that *nucleotide alignment* is a specialisation of *alignment tool* and *identified sequence* is an aggregation of *sequence* and *identification*.

The Domain Ontology also provides semantic annotation for workflows and data. This is exemplified in Section 4.3.

4.2 Planning with ontologies

A SHOP2 specification (Nau et al., 2003) is composed of three sections: domain definition (*defdomain*); problem definition, defining the problem that the planner must solve (*defproblem*); and requests to find the plans that solve a given problem (*find-plans*).

SHOP2 domain definition requires *methods* and *operators*. Operators specify available activities to implement methods. Methods are used in the planning process – a plan, at its highest level, is a concatenation of method invocations, recursively refined into methods and operators. Methods provide a convenient way to write problem-solving 'recipes' that are used by SHOP2 in order to solve problems, they correspond to activity types in our Service Ontology or workflows in the Workflow Repository. Figure 5 shows part of the translation of our Bioinformatics Domain and Service Ontologies into SHOP2. This translation is automatic and performed by the Translator module.

From a high level point of view, our translator produces a *defdomain* specification enriched with concepts from the Ontology Repository. Concepts in the Service Ontology and relationships in the Domain Ontology are translated into operators. Service and Domain concepts are subsequently translated into methods. Translation works as follows. We now explain in detail how ontologies are introduced to help planning (i.e., workflow construction).

Each 'type of service' concept in our Bioinformatics Service Ontology generates a SHOP2 operator whose pre-conditions are the service's input data types and whose post-conditions are the output data types. For example, Lines 2, 3, 6 and 7 of Figure 5, respectively, show the service types *ORFFinder*, *nucleotide alignment*, *filter* and *base calling* translated into SHOP2 operators. In more detail, *nucleotide alignment* is a type of service described in SHOP2 (Line 3, Figure 5) as an operator whose pre-condition is the existence of an *identified sequence* (input) and whose post-condition is the production of a *nucleotide alignment* (output). The value of 50.0 for this line indicates that the user provided this value as an upper bound estimate for the execution time of any nucleotide alignment service. As will be seen later on, the Evaluator will suggest instantiations for each service type, and use these values to rank suggestions.

A SHOP2 planner does not support generalisation/specialisation hierarchies of operators (tasks), neither does it manipulate complex objects. Thus, we had to extend this planner to fully support both facilities. To allow complex objects, the Translator creates SHOP2 operators that are used to represent ontological relationships. Complex objects are structures created from basic objects (basic concepts in our Domain Ontology). Since our ontologies contain aggregation and specialisation relationships, we use this knowledge to improve SHOP2 functionalities. For each aggregation relationship, we created an operation called *Compose*, with cost zero. Its input specifies the concepts that will be aggregated and its output is the aggregated concept (e.g., in Line 5 of Figure 5, an *identified sequence* aggregates a *nucleotide sequence* and its

identification). Similarly, we created the converse *Decompose* (e.g., Line 4 of Figure 5). To represent specialisations/generalisations, for each concept that is a specialisation, we create an operation called *IsSpecialisationOf* with cost zero that, given the specialised concept, returns the corresponding general concept. Similarly, we created the converse *IsGeneralisationOf*. These four operators extend SHOP2's planning capabilities. Whereas the other operators are transformed into service invocations by the Evaluator, these four serve only to help compose complex tasks in a plan.

Figure 5 Part of the SHOP2 definition domain for bioinformatics ontology

```
1  (defdomain bioinformatics-ontology (
2    (:operator (!orfFinder ?a) (sequence ?a) () ((ORFS ?a)) 25.0)
3    (:operator (!nucleotide_alignment ?a) ((identified_sequence ?a))
()((nucleotide_alignment ?a)) 50.0)
4    (:operator (!decompose_identified_sequence ?a) ((identified_sequence ?a)) ()
((nucleotide_sequence ?a) (identification ?a)) 0.0)
5    (:operator (!compose_identified_sequence ?a) ((nucleotide_sequence ?a)
(identification ?a) (not(identified_sequence ?a))) () ((identified_sequence ?a)) 0.0)
6    (:operator (!filter ?a) ((sequence ?a) (not(filtered ?a))) () ((filtered ?a)) 4.0)
7    (:operator (!base_calling ?a) ((chromatogram ?a))()((identified_sequence ?a)) 10.0)
8
9    (:method (TO_identified_sequence ?x)
10     ((chromatogram ?x))
11     ((!base_calling ?x))
12     ((nucleotide_sequence ?x)(identification ?x))
13     ((compose_identified_sequence ?x))
14   )
15
16   (:method (find_orfs ?x)
17     ((nucleotide_sequence ?x))
18     ((!orfFinder ?x))
19   )
20
21   (:method (TO_alignment ?x)
22     ()
23     ((TO_nucleotide_alignment ?x))
24     ()
25     ((TO_aminoacid_alignment ?x))
26   )
27
28   (:method (TO_nucleotide_alignment ?x)
29     ((identified_sequence ?x))
30     ((!nucleotide_alignment ?x))
31     ( not (nucleotide_sequence))
32     ((:ordered((TO_identified_sequence ?x) (!nucleotide_alignment ?x))))
33   )
34
35   (:method (TO_aminoacid_alignment ?x)
36     ((aminoacid_sequence ?x))
37     ((!aminoacid_alignment ?x))
38     ( not (aminoacid_sequence))
39     ((:ordered((TO_aminoacid_sequence ?x) (!aminoacid_alignment ?x))))
40   )
41 )
42 )
```

Once all operators have been specified, methods can be defined. For each type of service in the Service Ontology, the Translator creates a SHOP2 method that describes which operator (or set of operators) can execute this method. For example, Lines 21, 28 and 35 of Figure 5, respectively, show the headers of the methods *alignment*, *nucleotide alignment* and *aminoacid alignment*.

In bioinformatics, many times, the user needs to obtain a certain result without knowing what tasks produce this result (or concept) – for example, starting from a chromatogram (input concept), he/she wants to obtain an alignment (output concept). To allow the user to pose such requests, the Translator needs to create additional methods whose execution will generate concepts in the Domain Ontology.

4.3 Creating and instantiating plans

The SHOP2 *problem definition* section (see Figure 6(A)) is composed by a problem name (e.g., problem1), the label of the definition domain (e.g., bioinformatics ontology), the state of the world (a set of conditions that are true in a given instant) and the set of methods that must be utilised by the planner (e.g., *nucleotide alignment ch1*). The SHOP2 *request to find plans* (see Figure 6(B)), identifies the problem to be solved and requirements for the solution plan (such as maximum cost).

Figure 6 Plan synthesis and selection

A. SHOP2 input problem

```
(defproblem problem1 bioinformatics-ontology
  ((chromatogram ch1))
  ((nucleotide_alignment ch1)))
```

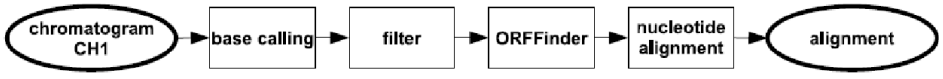
B. SHOP2 request to find plans

```
(find-plans 'problem1 :verbose :plans :which :all :optimise-cost 200)
```

C. SHOP2 output

```
Problem PROBLEM1 with :WHICH = :ALL, :VERBOSE = :PLANS, OPTIMISE-COST = 200
Totals: Plans Mincost Maxcost Expansions Inferences  CPU time  Real time
        3    60.0    85.0        29        70     0.017     0.042

Plans:
(
 ((!BASE_CALLING CH1) (!NUCLEOTIDE_ALIGNMENT CH1))

 ((!BASE_CALLING CH1) (!FILTER CH1) (!NUCLEOTIDE_ALIGNMENT CH1))

 ((!BASE_CALLING CH1) (!DECOMPOSE_IDENTIFIED_SEQUENCE CH1) (!ORFFINDER CH1)
       (!COMPOSE_IDENTIFIED_SEQUENCE CH1) (!NUCLEOTIDE_ALIGNMENT CH1))

 ((!BASE_CALLING CH1) (!FILTER CH1) (!DECOMPOSE_IDENTIFIED_SEQUENCE CH1)
       (!ORFFINDER CH1) (!COMPOSE_IDENTIFIED_SEQUENCE CH1)
            (!NUCLEOTIDE_ALIGNMENT CH1))
)
```

D. Workflow – graphical representation of the chosen plan



4.3.1 Plan specification

We now clarify the process of plan specification using an example. The user requests a plan to transform a *chromatogram* (input concept) into a *nucleotide alignment* (desired concept). Part A of Figure 6 shows the input request (already translated to SHOP2), indicating that the user wants the system to produce a workflow for a *nucleotide alignment* using *chromatogram ch1*. Notice that the user defines the request, but does not need to specify how to accomplish it. Part C of this figure shows the four possible plans (abstract workflows) generated by SHOP2 to answer the request, ordered by increasing cost order.

Plans are sets of methods that must be executed sequentially to achieve the goal. For instance, the first plan *((!BASE CALLING CH1) (!NUCLEOTIDE ALIGNMENT CH1))* shows that a possible solution is to execute a service of type *base calling*, using *ch1* as input and passing the result of this execution to a service of type *nucleotide alignment*. Symbolic variable *ch1* is a chromatogram as defined in the *defproblem* section – Figure 6(A).

This figure also shows, for instance, (Plans 3 and 4) the need for ontological concept manipulation. These plans chose the *ORFFinder* service type, which needs the *sequence* concept as input (see Line 2 of Figure 5). However, the problem's input is a complex data object of type *identified sequence* (see Figure 4). Thus, in order to feed it to *ORFFinder*, the planner had to use our *Decompose identified sequence* and *Compose identified sequence* ontological relationship operators to, respectively, disaggregate the problem's input and re-aggregate the output from *ORFFinder*.

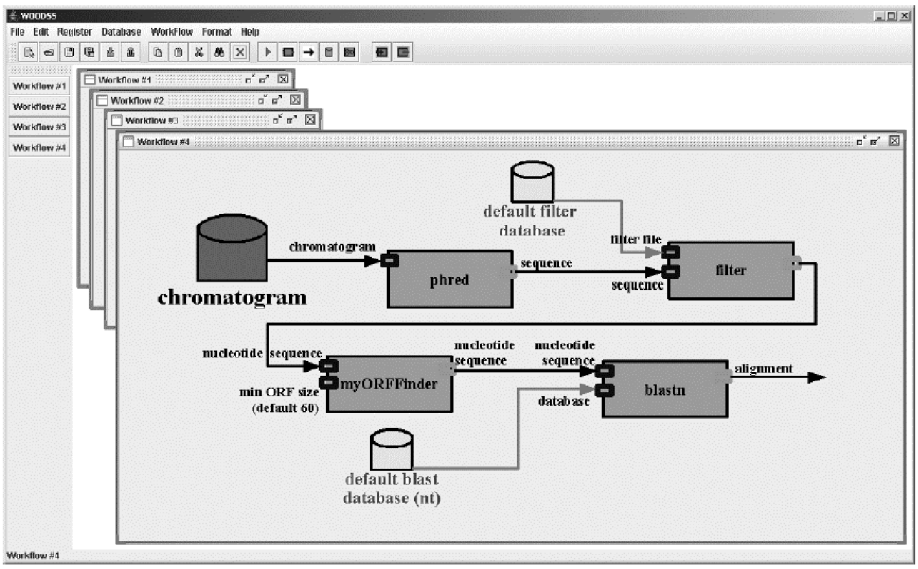
4.3.2 Plan instantiation

The four plans generated are abstract. For instance, a *nucleotide alignment* can be implemented by services *blastn* or by *FASTA*. Abstract plans are forwarded to the Evaluator that will generate an executable workflow by finding the web services that best fit each abstract service in the plan, using the ontological annotations of the Service Catalog. Our Evaluator takes several criteria (metadata from the Service Catalog) into consideration to select the most appropriate plan. These characteristics are retrieved by the Evaluator from the Service Catalog, together with service description. Figure 6(D) shows that the Evaluator chose the fourth plan. This plan was chosen because attributes in the service Catalog state that the use of the *filter service* increases the quality of the plan results and, if the user is looking for genes, the use of the *ORFFinder service* will improve the result.

Part D of Figure 6 shows a graphical version of the workflow for the selected abstract plan. Figure 7 shows a concrete (executable) version of this abstract workflow, using our graphical interface. Activities are rectangles, transitions are arrows and data repositories are represented as cylinders. Through this interface, the user can create, edit, annotate and execute workflows.

In manual editing, to insert a new activity in a workflow, the user must select one activity from the list of available activities from the Service Catalog. The system checks the consistency of all transitions (linking outputs of one activity to the inputs of another activity). Inputs can have default values (e.g., '60' is the default value for *minimum ORF size* of the *myORFFinder* activity).

Figure 7 Workflow graphical representation adopted from WOODSS



To insert new data, the user must select a data type from the list of available concepts of the Domain Ontology. Other operations allowed by the graphical interface are: embed a (sub) workflow inside a workflow, save, load, export and import workflows. This allows scientists to reuse (parts of) stored workflows to create new experiments.

The user can also monitor workflow execution. For instance, he/she can click in the arrows (transitions) to verify the produced data and the metadata associated (such as the name and version of the tool that produced this data; the date in which the data were produced; the origin of the data; etc.).

Scientists can designate which annotated workflows can be stored in the repository, to be used in subsequent experiments or to be shared with other scientists. When an experiment is saved, the data produced in each step of workflow execution are also saved. Moreover, a set of annotations are assigned to each produced data set, taking advantage of ontology type and service definitions. These annotations describe the origins of the data and the service (or set of services) that produced that data, facilitating data and service provenance traceability. In the example of Figure 7, the user, when consulting the result alignment, sees that the alignment was produced by *blastn service*, using the *blast database nt* and, as input sequence, the *ORF* produced by *myORFFinder service*.

5 Conclusions and ongoing work

This paper presented an ontology-based framework to support bioinformatics work. It helps the user in the three kinds of composition: manual, iterative and automatic. It takes advantage of AI planning techniques, combined with ontologies and Semantic Web standards. The solution is based on repositories that store information on services and their characteristics, on service and domain ontologies and on workflows. In particular, ontology repositories are extensively used in enhancing plan generation with semantics and in helping users design better scientific workflows.

Several bioinformatics laboratories have reported the use of scientific workflows and of a workflow infrastructure to support their experiments – e.g., (Oinn et al., 2004). Our work extends these approaches in three main directions: first, its use of AI planning techniques to help design the 'best' workflow for a task; second, the use of ontologies to semantically support workflow construction, both in selecting tasks and in finding appropriate services; third, the use of these ontologies in annotation and thus help traceability. We have built a prototype to verify and validate our proposal, for bioinformatics problems, specifically for genome assembly and annotation.

Ongoing work concerns mechanisms for improving provenance and traceability support. We also intend to explore other extensions for plan synthesis – e.g., re-planning and plan repair. We also intend to extend our bioinformatics ontology to reach a wider context in bioinformatics, such as comparative genomics and metabolic pathways.

Acknowledgements

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

References

Agarwal, V., Chafle, G., Kumar, K.A., Mittal, S. and Srivastava, B. (2005) 'Synthy: a system for end to end composition of web services', *Journal of Web Semantics*, Vol. 3, pp.311–339.

Blythe, J. and Ambite, J.L. (2004) *Workshop on Planning and Scheduling for Web and Grid Services*, Whistler, British Columbia, Canada.

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*, Vol. 31, pp.56–64.

Cavalcanti, M.C., Targino, R., Baião, F., Rössle, S.C., Bisch, P.M., Pires, P.F., Campos, M.L.M. and Mattoso, M. (2005) 'Managing structural genomic workflows using web services', *Data and Knowledge Engineering*, Vol. 53, No. 1, pp.45–74.

da Costa, L.A.G., Pires, P.F. and Mattoso, M. (2004) 'Automatic composition of web services with contingency plans', *IEEE ICWS 2004, IEEE Computer Society*, San Diego, California, USA, pp.454–461.

Fileto, R. (2003) *The POESIA Approach for Services and Data Integration on the Semantic Web*, PhD Thesis, IC–UNICAMP, Campinas, SP.

Fileto, R., Medeiros, C.B., Liu, L., Pu, C. and Assad, E. (2003) 'Using domain ontologies to help track data provenance', *Proc. Brazilian Database Conference, SBBD*, Manaus, Amazonas, Brazil, pp.84–98.

Fujii, K. and Suda, T. (2004) 'Dynamic service composition using semantic information', *ICSOC 2004*, ACM Press, New York City, NY, USA, pp.39–48.

Gao, H.T., Hayes, J.H. and Cai, H. (2005) 'Integrating biological research through web services', *IEEE Computer*, Vol. 38, No. 3, pp.26–31.

Ghallab, M., Nau, D. and Traverso, P. (2004) *Automated Planning, Theory and Practice*, Elsevier, London, UK.

Kalfoglou, Y. and Schorlemmer, M. (2003) 'Ontology mapping: the state of the art', *KER*, Vol. 18, No. 1, pp.1–31.

Knublauch, H., Fergerson, R.W., Noy, N.F. and Musen, M.A. (2004) 'The protege OWL plugin: an open development environment for semantic web applications', *Third International Semantic Web Conference (ISWC2004)*, Hiroshima, Japan, pp.229–243.

Levesque, H.J., Reiter, R., Lesperance, Y., Lin, F. and Scherl, R.B. (1997) 'Golog: a logic programming language for dynamic domains', *Journal of Logic Programming*, Vol. 31, Nos. 1–3, pp.59–84.

Long, D. and Fox, M. (2003) 'The 3rd international planning competition: results and analysis', *Journal of Artificial Intelligence Research*, Vol. 20, pp.1–59.

McIlraith, S.A. and Son, T.C. (2002) 'Adapting golog for composition of semantic web services', *KR2002*, pp.482–493.

Medeiros, C.B., Perez-Alcazar, J., Digiampietri, L., Pastorello, G., Santanchè, A., Torres, R., Madeira, E. and Bacarin, E. (2005) 'WOODSS and the web: annotating and reusing scientific workflow', *ACM SIGMOD Record*, Vol. 34, No. 3, pp.18–23.

Medjahed, B., Bouguettaya, A. and Elmagarmid, A. (2003) 'Composing web services on the semantic web', *VLDB Journal*, Vol. 12, pp.333–351.

Munoz-Avila, H., Aha, D.W., Nau, D., Weber, R., Breslow, L. and Yaman, F. (2001) 'SiN: integrating case-based reasoning with task decomposition', *IJCAI 2001*, Morgan Kaufmann, Seattle, Washington, USA, pp.999–1004.

Nau, D., Au, T.C., Ilghami, O., Kuter, U., Murdock, W., Wu, D. and Yaman, F. (2003) 'SHOP2: an HTN planning system', *Journal of Artificial Intelligence Research*, Vol. 20, pp.379–404.

Nau, D., Gupta, S. and Regli, W. (1995) 'Artificial intelligence planning versus manufacturing-operation planning: a case study', *IJCAI 1995*, Morgan Kaufmann, Montreal, Quebec, Canada, pp.1670–1676.

Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M.R., Wipat, A. and Li, P. (2004) 'Taverna: a tool for the composition and enactment of bioinformatics workflows', *Bioinformatics*, Vol. 20, pp.3045–3054.

Rao, J. and Su, X. (2004) 'A survey of automated web service composition methods', *SWSWPC 2004*, Vol. 3387, pp.43–54.

Russel, S. and Norvig, P. (2003) *Artificial Intelligence: A Modern Approach*, Prentice-Hall, New Jersey, USA.

Sirin, E., Parsia, B., Wu, D., Hendler, J.A. and Nau, D.S. (2004), 'HTN planning for Web Service composition using SHOP2', *Journal of Web Semantics*, Vol. 1, No. 4, pp.377–396.

Smith, J.W. and Schulze-Kremer, S. (2003) 'The ontology of the gene ontology, in 'AMIA2003'. Service composition using SHOP2', *Journal of Web Semantics*, Vol. 1, No. 4, pp.377–396.

Srivastava, B. and Koehler, J. (2003) 'Web service composition – current solutions and open problems', *ICAPS 2003*, pp.28–35.

Stevens, R., Baker, P., Bechhofer, S., Ng, G., Jacoby, A., Paton, N.W., Goble, C.A. and Brass, A. (2000) 'TAMBIS: transparent access to multiple bioinformatics information sources', *Bioinformatics*, Vol. 16, No. 2, pp.184–186.

Stevens, R.D., Tipney, H.J., Wroe, C.J., Oinn, T.M., Senger, M., Lord, P.W., Goble, C.A., Brass, A. and Tassabehji, M. (2004) 'Exploring williams-beuren syndrome using mygrid', *Bioinformatics*, Vol. 20, Suppl. 1, pp.i303–310.

Traverso, P. and Pistore, M. (2004) 'Automated composition of semantic web services into executable processes', *Lecture Notes in Computer Science 3298*, pp.380–394.

Wainer, J., Weske, M., Vossen, G. and Medeiros, C.B. (1996) 'Scientific workflow systems', *Proc. the NSF Workshop on Workflow and Process Automation Information Systems*, Athenas, GA, USA, pp.1–5.

Yoo, I. and Hu, X. (2006) 'Biomedical ontology mesh improves document clustering qualify on medline articles: a comparison study', *cbms 0*, pp.577–582.

Yu, J. and Buyya, R. (2005) 'A taxonomy of scientific workflow systems for grid computing', *ACM SIGMOD Record*, Vol. 34, No. 3, pp.44–49.