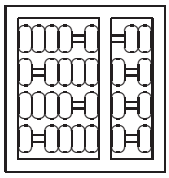

INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

A framework based on semantic Web services and AI planning for the management of bioinformatics scientific workflows

L.A. Digiampietri *J.J. Pérez-Alcázar*
C.B. Medeiros *J.C. Setubal*

Technical Report - IC-06-004 - Relatório Técnico

February - 2006 - Fevereiro

A framework based on semantic Web services and AI planning for the management of bioinformatics scientific workflows

Luciano Antonio Digiampietri*

José de Jésus Pérez Alcázar †and Claudia Bauzer Medeiros ‡

João Carlos Setubal §

Abstract

Bioinformatics activities are growing all over the world, with proliferation of data and tools. This brings new challenges, such as how to understand and organize these resources, how to exchange and reuse successful experimental procedures, tools and data, and how to provide interoperability among data and tools across different sites, and for distinct user profiles. This paper describes an effort towards these directions. It is based on combining research on databases, AI and scientific workflows, on the Semantic Web, to design, reuse, annotate and document bioinformatics experiments or parts thereof. The resulting framework allows the integration of heterogeneous data and tools, and the design of experiments as scientific workflows, which are stored in databases. Moreover, it takes advantage of the notion of planning in AI to support automatic or interactive composition of tasks. These ideas are being implemented in a prototype and validated on real bioinformatics data.

1 Introduction

Scientific workflows 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 applications [3].

Scientific workflows are being utilized in *in silico* experiments at the Laboratory for Bioinformatics (LBI) [25] at the University of Campinas, Brazil. LBI was the first Brazilian bioinformatics laboratory, being responsible for the coordination of the assembly and annotation of the *Xylella fastidiosa* genome [10]. The effort that led to the assembly and

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

†EACH, University of São Paulo, 03828-000, São Paulo, SP, Brazil, e-mail: jperez@usp.br

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

§Virginia Bioinformatics Institute, Virginia Tech, Bioinformatics 1, Box 0477, Blacksburg, VA, 24060, USA, e-mail: setubal@vbi.vt.edu

annotation of this genome involved the participation of over 30 laboratories in Brazil, each of which performed part of the tasks, while the integration and validation of all tasks was centered at LBI. Subsequently, the laboratory played the same role in other analogous efforts, devising efficient bioinformatics procedures involving agricultural genomes (e.g. sugar cane). This experience has naturally led to efforts in the specification and implementation of a computational framework for the management of bioinformatics scientific workflows [9].

Scientific workflows differ from business workflows in several points. In particular, in bioinformatics they are characterized by:

- high degree of human intervention: despite the existence of tools that support the execution of tasks, the majority of activities need a human curator to cheek the task flow and to validate the results;
- fluidity: because bioinformatics is still a new area, there is not a well-defined consensus about how the tasks must be executed and how the results must be annotated;
- opinion diversity: bioinformatics researchers need to be involved in the service selection process to make sure that their specialist opinions have been reflected in the obtained results.

In bioinformatics, furthermore, additional factors must be taken into consideration [28]. The workflows are usually designed manually, and activities can be defined using script languages or invoking Web services. The full acceptance of Web services technology depends greatly on the willingness of the biological research community to pursue standardization, including building ontologies, developing biology-specific registries, and defining the service interfaces for well-known functions [14].

Manual composition is a hard work and susceptible to errors, regardless of the use of Web services. Furthermore, in bioinformatics, due to the constant evolution of the area and the combinatorial explosion of alternatives, there are just too many alternatives for workflow construction. Thus, there is a pressing need for means to help scientists to design appropriate workflows.

The main idea behind our proposal is that the problem of automatic or semi-automatic composition of workflow tasks can be seen as an Artificial Intelligence planning problem. This approach has become interesting due to the maturity that the planning area has achieved in AI [27]. The paper attacks the problem of constructing scientific workflows, under the assumption that they are the basis for specifying and executing tasks in a distributed laboratory environment. The execution of each activity within such a workflow can be executed either by invocation of a Web service or of another (sub)workflow. Thus we use the terms “service composition” and “workflow composition/construction” interchangeably.

Our main contributions are:

- proposing a solution to the problem of composition of services combining results from AI and database systems, thereby helping design scientific workflows, while at the same time documenting design alternatives;
- validating the proposal by means of a prototype for genome assembly and annotation.

Our solution is based on a generic architecture whose core covers two issues: workflow construction and knowledge management. The first functionality is provided by a set of modules that produce alternatives for composing services using AI planning algorithms. Knowledge management is based on a set of repositories that store domain and services information, annotated workflows, and pieces thereof. The planning algorithms use the semantic information provided by the repositories to construct alternative workflows to perform a given task. Our architecture is specified in a generic way, and thus can be utilized to solve any problem that involves the storage, coordinated execution and automatic composition of scientific processes.

Our implementation takes advantage of WOODSS (WOrkflOw-based spatial Decision Support System) [32], a scientific workflow infrastructure developed at University of Campinas, Brazil. Originally conceived for decision support in environmental planning, it has evolved to an extensible database-centered environment that supports specification, reuse and annotation of scientific workflows and their components.

The rest of this paper is organized as follows. Section 2 describes related work and concepts. Section 3 describes AI planning solutions that can be exploited in supporting workflow composition. Section 4 presents the proposed architecture. Section 5 describes our prototype. Section 6 contains conclusions and ongoing work.

2 Related work and concepts

2.1 Genome assembly and annotation

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 so-called 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 assembly is a technological problem. Using
present technology, a genome has to be broken up in small pieces in order to be sequenced. For example, a bacterial genome has typically a few millions of base pairs, while the biggest piece of DNA that can be sequenced in a laboratory has about one thousand base pairs. Genome assembly is however just the beginning.

Genome annotation, that is, the assignment of functions to each gene, is not a technological problem. For each new sequenced genome, there will probably be new genes (unknown until the moment) and genes already known for which it is necessary to assign a function. The empiric verification of gene functions is a time- and money-consuming activity. Considering that a bacterial genome has a few thousand genes, this verification becomes impracticable. Thus, most functions are assigned based in 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 by several activities [9]. These activities are typically complex, involving interactions among several basic tasks, human intervention

and access to heterogeneous data sources. It is common, in bioinformatics laboratories, to find pipelines that execute a subset of activities that compose the assembly and annotation process. Another common procedure in these laboratories is the creation of scripts to execute specific workflows involved in these activities. Each experiment is a workflow and each scientist or team of scientists develop their own workflows or pipelines to help their daily activities. However, this practice has little flexibility, hampering the edition and reuse of these pipelines or workflows.

Another problem is to find tools on the Web that execute some desired activity. This search is typically based on keyword queries that can 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.

An important issue is tool/task composition. A common means to do this is via scientific workflows, where activities imply the execution of tools or other workflows. We highlight three kinds of composition: manual, iterative and automatic. In a manual composition, the scientist chooses each activity and specifies the links between activities involved in the workflow. The workflow reflects the user's knowledge, in particular, the knowledge of each tool to be used, but this system is susceptible to errors because the scientist can link tools with incompatible interfaces or try to execute tools without satisfying pre-requisites. Iterative composition follows abstraction levels. The scientist specifies the workflow abstractly and, in each iteration, will refine the specification until achieving an executable workflow. In an automatic composition, the scientist expresses a request (of goals or tasks to be executed) and the system, automatically, designs a workflow from the available components. Even automatic composition may require user interaction for additional information in the composition and execution of activities.

2.2 Workflows and Web services in bioinformatics

A workflow denotes the controlled execution of multiple tasks in an environment of distributed processing elements. Workflows represent a set of activities to be executed, their interdependencies, inputs and outputs[43]. Scientific workflows differ from usual workflows because they have additional characteristics like high degree of flexibility, uncertainty and presence of exceptions. Moreover, a scientific workflow is often not completely defined before it starts. The scientist performs some tasks and decides on further steps only after evaluating the previous ones[49].

Scientific workflows are being increasingly used in conjunction with the specification and execution of various tasks on the Web, with wide acceptance in bioinformatics [3, 20, 32]. To support interoperability among sites, many activities invoked by such workflows are encapsulated into Web services.

An important characteristic required for such Web service based applications is the ability to select and integrate, in runtime and efficiently, heterogeneous services. This has led to the development of Web service composition languages such as WSBPEL and WSCI. The problem with these proposals is that the definition of new processes, which interact with already existing ones, must be done manually.

The Semantic Web has been proposed to solve such problems. However, this will require extending the languages to add semantics to service description and discovery - e.g., using ontologies. To achieve the goals of interoperability between heterogeneous systems, the development of Semantic Web services must address the following challenges:

- Automatic Discovery of Web Services: location, by software agents, of Web services offering a given service.
- Automatic Web Services Invocation: execution of a Web service previously identified by another program or agent.
- Automatic Composition of Web Services: selection, composition and operation of Web services, to accomplish a task, given a high level description of some goal.
- Automatic Monitoring of Web Services: follow up of the execution of the Web service, in order to know its status, and if any unexpected problems occurred.

Our work is concerned with scientific workflow design, and thus the composition of Web services.

2.3 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; it includes the automatic selection and inter-operation of Web services. Automatic composition has an important role in enabling the Semantic Web [1, 31]. Users 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 can 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 [40]. In the first case, the workflow is specified manually and only the selection and binding of atomic Web services with the workflow is done automatically. On the other hand, dynamic composition automatically creates the workflow and selects atomic services. As examples of dynamic workflow generation we can mention [8, 13].

The task of presenting a sequence of actions to achieve an objective is called in Artificial Intelligence *plan synthesis*, or *planning* [16, 42]. Planning is a mature area in AI, with well-studied algorithms. This research has been motivated, among others, by studies of search in space states, theorem proving and control theory, to satisfy the needs of robotics, scheduling and other domains. Such techniques are currently used in mobile robots, manufacturing processes, satellite control, emergency management, among others [34, 36].

Recent research efforts have investigated the use of planning to solve the problem of automatic composition of Web services [19]. Usually, a planning problem can be conceived as follows. Given a description of the world, an initial state S_o, a goal description S_g, and a

set of actions A which can alter the state of the world, we must find an action composition which transforms the initial state S_o into a final state S_g that satisfies the goal. Transposing this to the Web service realm, S_o and S_g are the initial state and goal specified by the user who requests the service; and A is the set of services available, that must be composed to meet the user's requirements.

According to [45], in order to use planning in the automatic composition of Web services, AI planning concepts must be extended to consider the following characteristics:

- Plans need complex control structures with loops, non-determinism and conditionals. Furthermore, unlike most AI planning approaches, the explicit definition of pre- and post-conditions is not always possible.
- The objects managed by Web services must be typed (with complex structure and description).
- Web services produce new objects at execution time that can be used by other Web services. In planning, on the other hand, it is assumed that all kinds of domain objects are known in the initial state.

We highlight other important characteristics, not usually found in AI planning, such as:

- Ranking: when there are several plans with similar functionalities, the use of non functional attributes, like cost or quality, can facilitate the choice of the plan most adequate to the user's needs.
- Abstractions in plans: plans need to support semantic constructions such as hierarchies, as well as compatibility with the different Semantic Web service description standards, like OWL-S (www.daml.org/services) and WSMO (www.wsmo.org).
- Definition of extended goals: besides stating the final state, goals can involve complex conditions on process behavior, and thus their evaluation may require additional mechanisms, e.g., do not reserve hotel until flight has been reserved.

Still other characteristics needed in service composition but not found in planning include concurrency in service access, use of Web standards and scalability.

3 AI Planning applied to services

This section discusses classes of planning systems and proposals that can be considered to compose Web services. We compare these proposals to justify our scientific workflow construction strategy. This is not an exhaustive list, there being a great number of recent proposals [19, 21, 44, 47].

3.1 Proposals using Golog

Golog [26] is a language based in situation calculus[41] and supports the specification and execution of complex actions in dynamic systems. McIlraith and Son [30] adopted and

extended Golog to allow the automatic building of Web services. The authors treat the Web service composition problem through generic procedures (predefined plan templates) and restrictions customized by the user.

The general idea is that software agents can reason using Web services to discover, execute, compose and interoperate automatically with Web services. The request is a generic procedure, and user restrictions can be expressed through extended Golog. The authors represent Web services as primitive or complex actions. Primitive actions can change the state of world or provide information that changes the knowledge state of the agent. Procedural constructs (if-then-else, while, etc) are used to create the complex actions (complex services). The knowledge base provides a codification, in Golog, of the pre-conditions and effects. The agents use a mix of procedural language constructions and concepts of first order logic (primitive services and restrictions).

3.2 PDDL based proposals

The great interest of the AI Planning community in Web service composition can be explained by the similarity between the notations used in OWL-S and PDDL (Planning Domain Definition Language). PDDL is a widely used formal language created by McDermott [15] to describe different kinds of planning problems (e.g., [29, 38]). OWL-S descriptions can be transformed into PDDL.

McDermott [29] uses PDDL to specify plans, but he extends the language to introduce a new kind of knowledge, called value of the action, that represents the passage of information among the steps of the plan. This facilitates to distinguish information transformation and from state change. The planner used is a regression based, and allows the generation of conditional plans with ramifications. It is necessary when using Web services in a non-deterministic environment.

3.3 Rule based planning

Rule based planning methods, as their name indicates, use rules to represent actions and to specify plan generation.

Medjahed et al [33] propose to compose services starting from a high level declarative description, using composition rules to determine if two services can interact (composable). The composition process consists of four stages: specification, matchmaking, selection and generation. The specification stage consists of a high level description of the desired compositions using a XML based language. The matchmaking stage uses the compositions rules to generate the possible composition plans that represent the specification of the service required by the user. Selection selects the best plans in agreement with the user's suggestions. The final stage, generation, automatically generates the detailed description of the composite service and shows it to the user. The main contribution of this method is the use of composition rules that can be used as guides to other methods based in planning.

SWORD [39] is another rule based method. Services, modeled through pre- and post-conditions, are specified in a model of the world that consists of entities and relationships among the entities. A Web service is represented in the form of a Horn rule that describes

the post-conditions that are reached if the pre-conditions are true. To create a composite service, the user needs to specify only the initial and final states of the desired service. Plan generation can then be done by a rule-based system.

3.4 Hierarchical planning

Hierarchical planning is an AI planning methodology that creates plans by task decomposition. One well-known hierarchical planner is SHOP2 (Simple Hierarchical Ordered Planner 2) [35] which is based on Hierarchical Task Network (HTN) [42]. SHOP2 won the prize of one of the four best planners in the 2002 International Planning Competition [27]. Sirin et al [44] use SHOP2 for the automatic composition of Web services. 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 ontology. One of the problems of this proposal is the fact that it does not allow the generation of plans with concurrent control structures (i.e. Split and Split+Join). These kinds of structure are useful in the composition of Web services and are part of the structure of OWL-S composite processes.

SHOP2 uses the concept of methods to decompose a task in sub-tasks. These methods can contain explicit actions for monitoring that allow the planner to obtain the necessary data to treat problems of incomplete information. SHOP2 can implement a kind of extended goal through the specification of composite tasks (methods) that describe the changes required by the users. SHOP2 has demonstrated good results when using a great amount of methods and operators (simple tasks). It was, recently, extended to deal with non-determinism (ND-SHOP2) [22].

3.5 Symbolic Model Checking (SMC) based planning

Model Checking is a formal method often used in verification of complex hardware and software systems [6]. Symbolic Model Checking (SMC) is a particular form of Model Checking that allows to analyze large finite-state systems by means of symbolic representation techniques [2]. More recently, this technique has been applied to planning with remarkable success. The planning as model checking approach [18]formulates a planning problem in a logical context, while the symbolic representation techniques allow for handling complex domains.

The work of Traverso and Pistore [47], shows a technique for the automatic composition of Web services described in OWL-S that allows the automatic generation of executable processes (in WSBPEL). The goals that specify the services to be automatically generated are represented in the Eagle language [24]. This language has a clear semantic that can express complex requirements.

The authors use a planning approach based in a symbolic model checking technique [5], called MBP (Model Based Planner). This technique has presented good practical results for the problem of planning with non-deterministic actions, partial observations (the environment is not fully known), complex goals and domains (very large space of states). One of its problems is that it does not explore the hierarchical and taxonomical aspects of OWL-S.

3.6 Comparison of proposals

Table 1 summarizes the comparison of the proposals. The first column contains the requirements for planning using Web services that we pointed out in Section 2.3. The other columns cover the main classes of planning systems reviewed.

	Golog	PDDL	SWORD	Medjahed et al.	SHOP2	MBP
Use of standard	Y	Y	N	Y	Y	Y
Complex objects	N	Y	Y	Y	N	N
Abstraction / Hierarchy	N	N	N	N	Y	N
Non-determinism / Partial obs. of the world	Y	Y	N	Not clear.	Y	Y
Generation of non-linear plans	Y	Y; partially	N	Not clear.	Y	Y
Automation level	SA	A	A	A	A	A
Plan selection	N	N	N	Y	N	N
Concurrency	Y	N	N	Y	N	Y
Scalability	NM	NM	W	NM	G	G
Extended goals	Y	N	N	N	Y	Y

N → No; Y → Yes, SA → Semi-automatic; A → Automatic
NM → Not mentioned; W → Weak; G → Good.

Table 1: Comparison among planner proposals

Except for SWORD, all analyzed proposals use standards like WSDL or OWL-S. The proposal based in Golog implements support for non-deterministic and partial observations through predefined generic procedures. These procedures guarantee a certain level of non-determinism that is solved at the moment of plan generation. McDermott's planning based in PDDL gives this guarantee through sensing operations and conditions. However, the best solution is the proposal using SMC because it is more general. Other proposals, such as those of Medjahed and SHOP2 do not mention non-determinism. None of these proposals mentions dynamic object creation.

We also observe in Table 1 that none of the reviewed solutions covers all issues stated in column 1. Moreover, planning using Web services is a recent research area (2002) and the majority of these projects are at an experimental stage. The works on SHOP2 and SMC (MBP) seem to be the best alternatives. We decided to use, SHOP2 in our implementation, because it provides the following benefits: (a) it enables embedding domain knowledge to control the search space and improve efficiency; (b) it has been successfully used in a variety of real-world planning-based applications; (c) it allows inclusion of different types of precondition constraints for service operators as well as calls to external systems; (d) it enables modeling process abstractions in terms of method/operator hierarchies; (e) it enables reuse by facilitating selection of appropriate methods from domain-related operator libraries.

4 An architecture for automatic composition via planning

This section outlines our general architecture for composition of workflows and Web services. From the comparison of the previous section we note that none of the analyzed proposals treat complex objects or objects created dynamically, two very important characteristics within Web services. However, planning algorithms can be expanded to work with expressive representation of knowledge, e.g. ontologies [17]. Our proposal integrates all these concepts. The present implementation uses the SHOP2 algorithm.

4.1 Generic Composition Architecture

Figure 1 summarizes our composition architecture that extends the framework defined by Rao and Su [40]. Domain and service ontologies respectively store information on application domain and services available. The core of the architecture consists of the planner and the ontologies.

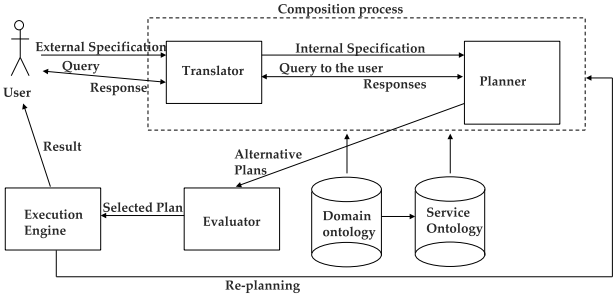

Figure 1: Architecture for automatic composition and execution of services

The Web service composition and execution process starts when the user of the services (human or software agent) makes a request for a service, which is translated to the planner's internal specification language. In the present implementation (see Section 5) this language is the one used by SHOP2, an extension of Lisp. The request for a service can be the description of a goal or task; starting from it, the planner generates alternative execution 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. The planner can, during the composition process, query the user or available services, to solve problems of incomplete information and non-deterministic domain behavior. Once the plans are generated, they are passed on to the evaluator, which chooses the best plan to meet the user needs. The executor is responsible for the execution of the chosen plan, forwarding the results to the user.

Domain and service ontologies, stored in ontology bases, are key to this process. In more detail, the *translator* module uses these bases to transform a user's request into a plan specification language, in our case, an extension of Lisp.

The *planner* module solves these requests, composing the services published by the service providers. It accesses the ontological bases to obtain the functionalities of the services and generates a process model (in our case, scientific workflows) that describes

the composite service. The planner uses the domain ontology to improve the efficiency in the planning process and to facilitate the modeling and the management of complex objects. The planner's output contains several workflows (the plans) with equivalent or similar functionalities.

The *evaluator* module uses non-functional service attributes (execution time, quality, trust, etc) and the knowledge about the user to select the best alternative among the plans generated by the planner. The user can specify weights to each non-functional attribute and the best composite service is the one that is ranked on top. The *executor* is responsible for the execution of the chosen plan. Due to the non-determinism and the partial observation of the world, the executor can interact with the planner in the case of failures.

The next section describes how this composition and execution architecture is integrated into our general refined architecture.

4.2 Refined Architecture

Our plans are specifications of scientific workflows. Thus, in this section, we will use indistinctly both terms. We start from the planning framework defined in Section 4.1, and specify a more complex architecture, able to deal with automatic composition of workflows based in Web services. Figure 2 shows this refined architecture, highlighting the main modules and their interactions. It extends the WOODSS scientific workflow framework [32]. Whereas WOODSS is based on manual composition, our architecture supports automatic and semi-automatic composition via planning. Section 5 presents our implementation of part of this architecture.

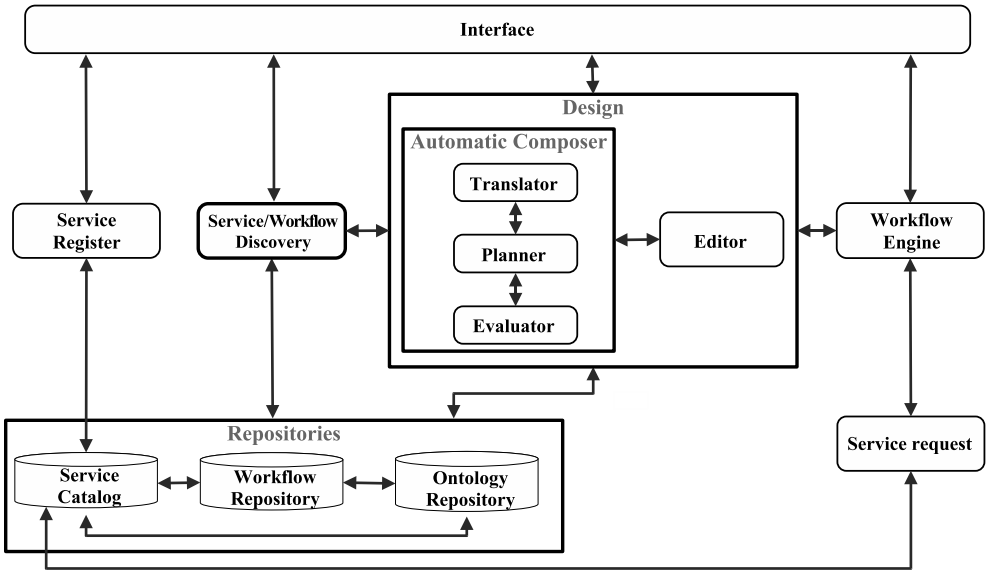

Figure 2: System Architecture

The architecture is based on the use of three repositories. The Ontology repository contains the domain and service ontologies of Figure 1 – in our case study, information about

genome assembly and annotation. The Service Catalog plays the role of a UDDI (Universal Description Discovery & Integration) registry, storing the profiles of services and workflows. The Workflow Repository, adopted from WOODSS, stores annotated (sub)workflows deemed important by the user (e.g., those used frequently).

The Interface Layer has a user-friendly graphical workflow specification and editing tool. It also allows the user to register services and workflows, request the execution of a workflow and interact with this execution.

The Service/Workflow Discovery module is responsible for the search of services and workflows that meet user requests. Our infrastructure allows search based on functionality, context and syntax. Search for functionality and context is based in the semantic data assigned to the services (metadata that associate the services with the ontologies). The search for syntactic compatibility is based on the parameters of the service interfaces. When no stored service or workflow meets the requests, this module will ask the Design module to create new workflows.

The architecture of Section 4.1 is represented in our general architecture through Automatic Composer and Workflow Engine. The Automatic Composer module encapsulates Translator, Planner and Evaluator. It receives a plan request and generates workflows automatically or semi-automatically. To generate these workflows, the Translator needs first to convert the request to the specific planner language. Next, the Planner interacts with the Service Catalog, Workflow Repository and the Ontology Repository to obtain information for plan generation. With these data, the Planner is able to generate workflows automatically answering the request from Workflow Discovery. The Evaluator chooses among the workflows (plans) generated, the workflow that best suits the request. This selection can be guided by the user. At execution time, the Workflow Engine module interacts with the Automatic Composer module. Whenever a fault is detected, the Workflow Engine module can ask the Automatic Composer module for the generation of an alternative plan that replaces the faulty service by an equivalent service or by a new workflow.

The Editor module has two main roles: workflow design and domain and service ontologies update. For workflow design, we used WOODSS graphical interface (see Figure 6) that access the workflow repository and lets the user manually compose, reuse and annotate workflows. Annotations include free text and references to the ontology repository. Ontology editing uses basically Protégé
(http://protege.stanford.edu) which is being coupled with an implementation of hyperbolic trees to facilitate dynamic visualization [12, 48].

The user interacts with the Service Register module in order to define new services. These services are described in WSDL and OWL-S, thereby linked to the Ontology Repository.

The Workflow Engine module follows the specification of [7]. It is responsible for the execution control of all workflow activities via orchestration. These activities can be a simple Web service or a complex workflow. It corresponds to the Execution Engine of Figure 1. The operations provided by the Workflow Engine are: interpretation of the complex process definitions; creation and management of the process instances; and supervisor and management functions [7]. The module 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 like service unavailability or connection timeout.

This architecture supports the three kinds of composition presented in Section 2.1. In the manual composition, the system verifies the consistence between the inputs and outputs of the workflow's activities. In iterative composition, for each iteration, the system suggests to the user activities or sub-workflows that have been previously stored in the repositories and that can be used for an already defined task. In the automatic composition, it designs a set of workflows (solutions) that satisfy the conditions provided by the user.

This architecture can be used to solve any kind of problem of automatic composition of Web services. It can be specialized to work on different needs by populating the repositories with the data corresponding to a given domain. The next section discusses one particular implementation, for bioinformatics.

5 Case study: genome assembly and annotation

We implemented a prototype of the architecture presented in Section 4.2 to solve the problems of genome assembly and annotation discussed in Section 2.1.

Our system allows re-planning, supporting communication between the Planner and the Executor when there is a fault in the plan execution that cannot be solved through interaction with the user.

In order to implement the architecture we had to construct the appropriate ontologies. In particular, we have developed a detailed ontology, specific to genome assembly and annotation, that extends a generic bioinformatics ontology [46]. Through our ontology we annotated bioinformatics data and tools/services in order to allow semantic search and automatic composition of services. Figure 3 shows a portion of our ontology and its annotation of services. To simplify the figure, we omitted several relationships. Domain and Service portions of the ontology are separated, thus helping establish distinct relationships among the concepts.

As shown in Section 4, the Service Catalog contains the profiles of the services. These profiles are instances of services of the Service Ontology. The parameters (input and output) of the services are complex or simple concepts of the Domain Ontology. We highlight only the relationships involved with *nucleotide alignment*. The *blastn service* in the Service Catalog implements a *nucleotide alignment*, which is an *alignment tool* description. The *nucleotide alignment* has as input an *identified sequence* and as output an *alignment*, both concepts of the application domain. The dotted line between *nucleotide alignment* and *nucleotide sequence* indicates a restriction of the input data: the *sequence* of the *identified sequence* must be a *nucleotide sequence*.

In our ontology domain, all concepts are atomic data types (integer, strings, etc) or, recursively, an aggregation and/or a generalization (or specialization) of concepts. We can observe in Figure 3 that *nucleotide alignment* is a specialization of *alignment tool* and *identified sequence* is an aggregation of *sequence* and *identifier*. Our implementation uses

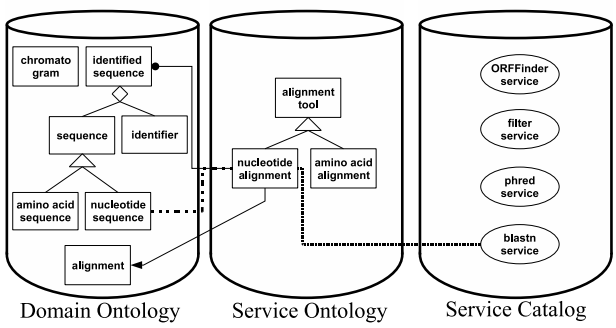

Figure 3: Small example of the relationships among our repositories

SHOP2 [35] - see section 3 for justification of this choice.

A SHOP2 specification is composed by three sections: domain definition (*defdomain*, see Figure 4); problem definition, defining the problems that the planner must solve (*defproblem*, see Figure 5A); and requests to find the plans that solve a given problem (*find-plans*).

SHOP2 *domain definition* is composed by *operators* that are the basic activities that change the state of world, and *methods* which explain how to decompose a compound task into a set of subtasks [35]. Figure 4 shows a small piece of our bioinformatics ontology, described as a SHOP2 domain definition. For each service in our Service Catalog we needed to create a SHOP2 operator whose pre-conditions are the service's input data and whose post-conditions are the output data and the changes on the state of the world. For example, lines 2, 3, 6 and 7 of Figure 4 show, respectively, the services *ORFFinder*, *blastn*, *filter* and *phred* (in the catalog of Figure 3). For instance, *blastn* is a *nucleotide alignment* tool/service (see annotation in Figure 3). It is described in SHOP2 (line 3, Figure 4) as an operator whose pre-condition is the existence of a *sequence* (input) and post-condition the production of a *nucleotide alignment*. The indication of 50.0 in the figure, for this line, denotes the execution time of this operator, a factor to be used by the evaluator when ordering plans.

Analogously, for each kind of service in the Service Ontology, we create a SHOP2 method that describes which operator (or a set of operators) can execute this method. For example, lines 21, 26 and 36 of Figure 4 show, respectively, the headers of the methods *alignment*, *nucleotide alignment* and *aminoacid alignment*.

SHOP2 does not support general semantic constructs - e.g., the aggregations and specializations of our ontology. Thus, we had to extend this planner. To allow complex objects, we created operations that denote relationships among the concepts. These operations are used only to represent ontological relationships in SHOP2. They do not appear in the workflows that are a result of planning strategies. For each aggregation, we created an operation in SHOP2 called *Compose* with cost zero. Its input specifies the aggregation components, its output the aggregated concept (e.g. line 5 of Figure 4, an *identified sequence* aggregates a *sequence* and its *identifier*). Similarly, we created the converse *Decompose* (e.g. line 4 of Figure 4). To represent specializations/generalizations, for each concept that is a specialization, we create an operation called *IsSpecializationOf* with cost zero that, given the specialized concept, returns the corresponding general concept.

The SHOP2 *problem definition* section (see Figure 5A) 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 utilized by the planner (e.g. nucleotide_alignment ch1). The SHOP2 *request to find a plan* identifies the problem that we want to solve and some requisites for the solution plan (like maximum cost).

As mentioned in Section 3.4, SHOP2 uses a task list as a goal. In bioinformatics, many times, the user's goals are to obtain a certain concept without knowing what tasks produce this concept - for example, starting from a chromatogram, he/she wants to obtain an alignment. To allow the user to request goals and tasks in the same way, we created a method for each kind of task that produces a concept in the domain ontology and added it to the domain definition section. When the user requests a plan to produce a concept, the system sends to the planner a request for the method that achieves the desired concept. SHOP2 will return plans that, starting from input concepts, use these methods to produce the desired concept.

Let us clarify this using an example, where the user requests a plan that transforms a *chromatogram* (input concept) into a *nucleotide alignment* (desired concept). In part A of Figure 5, we show the input of SHOP2 (translated to the SHOP2 specific language), corresponding to a request for a *nucleotide alignment* using *chromatogram ch1*. In part B of this figure, we show the four possible plans (workflows) generated to answer the request. It shows, for instance (plans 3 and 4) the need for ontological concept manipulation. These plans chose the *ORFFinder* service, which needs the *sequence* concept as input. However, the problem's input is ch1, and *identified sequence* - an aggregated concept. Thus, in order to feed it to *ORFFinder*, the planner had to specify the *Decompose_identified_sequence* and *Compose_identified_sequence* operators to, respectively, disaggregate ch1 and re-aggregate the output from *ORFFinder*.

Plans are ordered in processing cost. These plans are next passed on to the Evaluator. Besides cost, our plan evaluator considers other non-functional characteristics to select the most appropriate plan. All characteristics (processing cost, trust, quality, etc) are stored in the Service Catalog, together with service description. The Evaluator chose the fourth plan (depicted as a workflow in part C) because 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 C of Figure 5 shows a graphical version of the workflow that corresponds to the selected plan. Our workflows use the workflow model proposed by Pastorello et al [37]. Some advantages of this model are: it allows the representation of all kind of structure necessary to a scientific workflow (like loops, conditionals, etc) and there is a simple algorithm to convert its workflows to languages for service composition (like WSBPEL). We extended the WOODSS scientific workflow framework [32] to allow a detailed visualization and management of our workflows.

Figure 6 shows a screenshot of the WOODSS graphical interface used by us. In its graphical representation of workflows, 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. To insert a new activity in a workflow, the user must

select one activity from the list of available activities from Service Catalog. For example, *ORFFinder*, that has as input a *nucleotide sequence* and *minimum ORF size*; its output is an *ORF*, that is a specialization of a *sequence*. The system checks the consistence of all transitions (that link outputs of one activity to the inputs of another activity). Inputs can have default values (e.g. the *minimum ORF size* of the *ORFFinder* activity has as "60" as default value). To insert new data, the user must select the data type in the list of available concepts of the Domain Ontology. Other operations allowed by the graphical interface are: insert a (sub-)workflow inside a workflow, save, load, export and import workflows.

Following WOODSS philosophy, scientists can designate which (annotated) workflows can be stored in the repository, to be used in subsequent experiments or shared with other scientists.

6 Conclusions and ongoing work

We specified an architecture based on Web services that allows the integration of heterogeneous data and applications, and supports automatic or interactive service composition through the use of AI planning. We have built a prototype to verify and validate our proposal, for bioinformatics problems, specifically for genome assembly and annotation. The choice of this specific area was made due to our experience with these tasks and the great need of this kind of system in bioinformatics [11].

Our main contributions lie in proposing and prototyping a solution for specifying scientific workflows in the Web by taking advantage of AI planning techniques, combined with ontologies and Semantic Web standards. An associated contribution is the extension of the algorithm proposed by Sirin et al [44] to allow complex data types in the SHOP2 planner.

Our architecture is generic, and can be instantiated for several domains. It helps the user in the three kinds of composition: manual, iterative and automatic. Manual composition is very useful when the user knows exactly what activities he/she desires to compose. Iterative composition is advisable when the user has a general knowledge of the process that he wants execute, but does not know what tools/services execute this process. In this case, the system suggests the activities. Automatic composition is advisable when the user knows pre- and post-conditions, but does not know (or is not interested in) how to design a workflow that satisfies these conditions.

As future work we intend to explore other promising ways for plan synthesis as alternatives to SHOP2. For example, Kuter et al [23] propose an algorithm that combines the power of the strategy of search control of hierarchical planning (SHOP2) with planning techniques of symbolic model checking (MBP). We also intend to study the use of symbolic model checking and the possibility to combine this technique with HTN. Another important step in plan synthesis is the use of plan repair techniques to decrease the need for re-planning [4]. In bioinformatics there are several tools with similar functionality, and thus, the use of plan repair may be a good strategy. Finally, we have in mind to extend our bioinformatics ontology to reach a wider context in bioinformatics, such as comparative genomics and metabolic pathways.

7 Acknowledgments

The work described in this paper was partially financed by CAPES, the MCT-PRONEX SAI project and CNPq WebMaps and AgroFlow projects. JCS and CBM were funded by CNPq fellowships. JJPA was partially financed by the Autonomous University of Bucaramanga.

References

[1] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):28–37, 2001.

[2] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: $10\hat{2}0$ States and Beyond. *Information and Computation*, 98(2):142–170, 1992.

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

[4] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau. Using iterative repair to improve the responsiveness of planning and scheduling. In *AIPS2000*, pages 300–307, 2000.

[5] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artif. Intell.*, 147(1-2):35–84, 2003.

[6] E. M. Clarke and J. M. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, 1996.

[7] T. W. M. Coalition. Workflow management coalition terminology & glossary (issue 3.0) 1999. http://www.wfmc.org/standards/docs (as of 2005-09-20).

[8] L. A. G. da Costa, P. F. Pires, and M. Mattoso. Automatic Composition of Web Services with Contingency Plans. In *IEEE ICWS 2004*, pages 454–461. IEEE Computer Society, 2004.

[9] L. A. Digiampietri, C. B. Medeiros, and J. C. Setubal. A framework based in Web services orchestration bioinformatics workflow management. *Genetics and Molecular Research*, 4(3), 2005.

[10] A. J. G. S. et al. The genome sequence of the plant pathogen Xylella fastidiosa. *Nature*, 406(1):151–157, 2000.

[11] L. A. D. et al. Fact and Task Oriented System for genome assembly and annotation. *LNBI*, 2594:238–241, 2005.

[12] R. Fileto. *The POESIA Approach for Services and Data Integration On the Semantic Web*. PhD thesis, IC–UNICAMP, Campinas–SP, 2003.

[13] K. Fujii and T. Suda. Dynamic Service Composition Using Semantic Information. In *ICSOC 2004)*, pages 39–48. ACM Press, 2004.

[14] H. T. Gao, J. H. Hayes, and H. Cai. Integrating Biological Research through Web Services. *IEEE Computer*, 38(3):26–31, 2005.

[15] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL the planning domain definition language. In *Proc. of AIPS-98 Planning Committee*, 1998.

[16] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning, Theory and Practice*. Elsevier, 2004.

[17] Y. Gil. Description Logics and Planning. *AI Magazine*, 2005.

[18] F. Giunchiglia and P. Traverso. Planning as Model Checking. *Lecture Notes In Computer Science*, 1809:1–10, 1999.

[19] *Workshop on Planning and Scheduling for Web and Grid Services*, June 2004.

[20] R. B. J. Yu. A taxonomy of scientific workflow systems for grid computing. *ACM SIGMOD Record*, 34(3):44–49, 2005.

[21] J. Kim and Y. Gil. Towards Interactive Composition of Semantic Web Services. In *AAAI 2004*, march 2004.

[22] U. Kuter and D. Nau. Forward-Chaining Planning in Nondeterministic Domains. In *AAAI 2004*, pages 513–518, 2004.

[23] U. Kuter, D. Nau, M. Pistore, and P. Traverso. A Hierarchical Task-Network Planner based on Symbolic Model Checking. In *(ICAPS 2005)*, pages 300–310. AAAI Press, 2005.

[24] U. D. Lago, M. Pistore, and P. Traverso. Planning with a Language for Extended Goals. In *AAAI 2002*, pages 447–454. AAAI Press, 2002.

[25] Laboratory for Bioinformatics, Institute of Computing, University of Campinas. http://www.lbi.ic.unicamp.br (as of 2005-09-15).

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

[27] D. Long and M. Fox. The 3rd International Planning Competition: Results and Analysis. *Journal of Artificial Intelligence Research*, 20:1–59, 2003.

[28] P. Lord, S. Bechhofer, M. D. Wilkinson, G. Schiltz, D. Gessler, D. Hull, C. Goble, and L. Stein. Applying semantic web services to bioinformatics: Experiences gained, lessons learnt. In *Proc. of the the 3rd International Semantic Web Conference*, pages 350–364, november 2004.

[29] D. McDermott. Estimated-Regression Planning for Interactions with Web Services. In *AIPS 2001*, 2002.

[30] S. A. McIlraith and T. C. Son. Adapting Golog for Composition of Semantic Web Services. In *KR2002*, pages 482–493, 2002.

[31] S. A. McIlraith, T. C. Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

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

[33] B. Medjahed, A. Bouguettaya, and A. Elmagarmid. Composing web services on the semantic web. *VLDB Journal*, 12:333–351, 2003.

[34] H. Munoz-Avila, D. W. Aha, D. Nau, R. Weber, L. Breslow, and F. Yaman. SiN: Integrating case-based reasoning with task decomposition. In *IJCAI 2001*, pages 999–1004. Morgan Kaufmann, 2001.

[35] D. Nau, T. C. Au, O. Ilghami, U. Kuter, W. Murdock, D. Wu, and F. Yaman. SHOP2: An HTN Planning System. *Journal of Artificial Intelligence Research*, 20:379–404, 2003.

[36] D. Nau, S. Gupta, and W. Regli. Artificial Intelligence Planning versus manufacturing-operation planning: a case study. In *IJCAI 1995*, pages 1670–1676. Morgan Kaufmann, 1995.

[37] G. Z. Pastorello Jr. Publication and Integration of Scientific Workflows on the Web. Master's thesis, UNICAMP, 2005. In Portuguese.

[38] J. Peer. A PDDL Based Tool for Automatic Web Service Composition. *Lecture Notes in Computer Science*, 3208:149–163, 2004.

[39] S. R. Ponnekanti. Sword: A developer toolkit for web service composition. In *Proc. of 9th International World Wide Web Conference*, 2002.

[40] J. Rao and X. Su. A Survey of Automated Web Service Composition Methods. In *SWSWPC 2004*, volume 3387, pages 43–54, 2004.

[41] R. Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT press, 2001.

[42] S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.

[43] L. Seffino, C. B. Medeiros, J. Rocha, and B. Yi. WOODSS - A Spatial Decision Support System based on Workflows. *Decision Support Systems*, 27(1-2):105–123, 1999.

[44] E. Sirin, B. Parsia, D. Wu, J. A. Hendler, and D. S. Nau. HTN planning for Web Service composition using SHOP2. *Journal of Web Semantics*, 1(4):377–396, 2004.

[45] B. Srivastava and J. Koehler. Web Service Composition - Current Solutions and Open Problems. In *ICAPS 2003*, pages 28–35, June 2003.

[46] R. Stevens, P. Baker, S. Bechhofer, G. Ng, A. Jacoby, N. W. Paton, C. A. Goble, and A. Brass. TAMBIS: Transparent Access to Multiple Bioinformatics Information Sources. *Bioinformatics*, 16(2):184–186, 2000.

[47] P. Traverso and M. Pistore. Automated Composition of Semantic Web Services into Executable Processes. *Lecture Notes in Computer Science*, 3298:380–394, 2004.

[48] L. R. Venancio, R. Fileto, C. B. Medeiros, and E. Assad. Applying Geographic Object Ontologies to help Navigation in GIS (in Portuguese). In *Proc. of VI Brazilian Symposium on GeoInformatics*, November 2003.

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

```
(defdomain bioinformatics-ontology (
  (:operator (!orfFinder ?a) (sequence ?a) () ((ORFS ?a)) 25.0)
  (:operator (!blastn ?a) ((sequence ?a)) () ((nucleotide_alignment ?a)) 50.0)
  (:operator (!Decompose_identified_sequence ?a) ((identified_sequence ?a)) () ((sequence ?a)
(identification ?a)) 0.0)
  (:operator (!Compose_identified_sequence ?a) ((sequence ?a)(identification ?a)) ()
((identified_sequence ?a)) 0.0)
  (:operator (!filter ?a) ((sequence ?a) (not(filtered ?a))) () ((filtered ?a)) 0.0)
  (:operator (!phred ?a) ((chromatogram ?a))()((identified_sequence ?a)) 10.0)

  (:method (TO_identified_sequence ?x)
    ((chromatogram ?x))
    ((!phred ?x))
    ((sequence ?a) (identification ?a))
    ((!compose_identified_sequence ?a))
  )

  (:method (find_orfs ?x)
    ((sequence ?x))
    ((!orfFinder ?x))
  )

  (:method (TO_alignment ?x)
    ()
    ((TO_nucleotide_alignment ?x)
  )

  (:method (TO_nucleotide_alignment ?x)
    ((sequence ?x))
    ((!blastn ?x))
    ((not(sequence ?x)))
    ((TO_sequence ?x)(!blastn ?x))
  )

  (:method (TO_nucleotide_alignment ?x)
    ((ORFS ?x))
    ((!ORFS_IsSpecializationOf_sequence ?x)(!blastn ?x))
    ( not (ORFS ?x))
    ((:ordered((TO_orfs ?x)(!ORFS_IsSpecializationOf_sequence ?x)(!blastn ?x)))))
  )
)
```

Figure 4: Part of the SHOP2 definition domain for bioinformatics ontology

A. SHOP2 input problem

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

B. SHOP2 output

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

Plans:
(
 ((!PHRED CH1) (!BLASTN CH1))

 ((!PHRED CH1) (!FILTER CH1) (!BLASTN CH1))

 ((!PHRED CH1) (!DECOMPOSE_IDENTIFIED_SEQUENCE CH1) (!ORFFINDER CH1)
        (!COMPOSE_IDENTIFIED_SEQUENCE CH1) (!BLASTN CH1))

 ((!PHRED CH1) (!FILTER CH1) (!DECOMPOSE_IDENTIFIED_SEQUENCE CH1)
        (!ORFFINDER CH1) (!COMPOSE_IDENTIFIED_SEQUENCE CH1) (!BLASTN CH1))
)
```

C. Workflow – graphical representation of the chosen plan

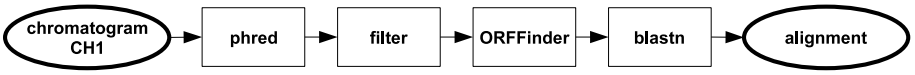

Figure 5: Plan synthesis and selection

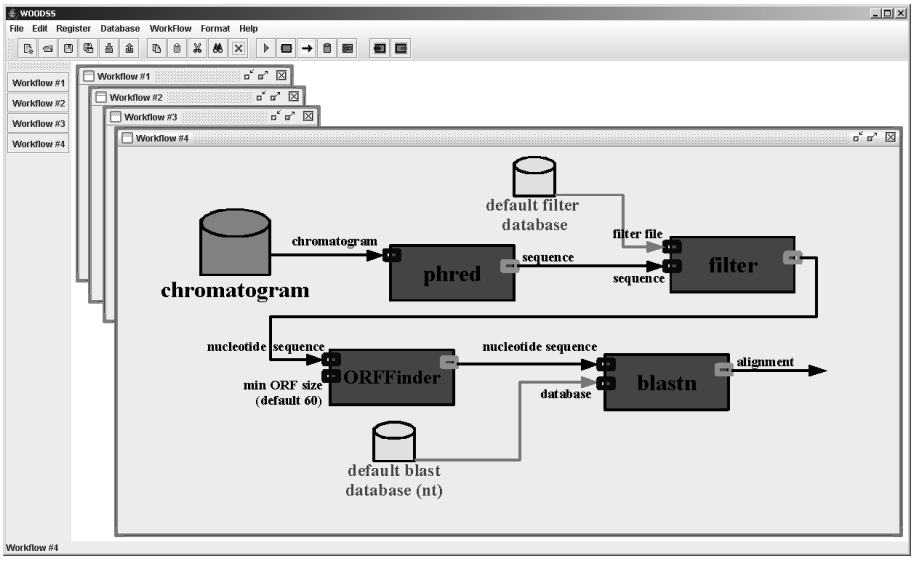

Figure 6: Workflow Graphical Representation in WOODSS