

Semi-Automatic Composition of Web Services for the Bioinformatics Domain

Zhiming Wang, Rui Wang, Cristina Aurrecochea, Douglas Brewer,

John A. Miller, Jessica C. Kissinger

University of Georgia, Athens, GA, 30602

{zhiming@cs., very@, aurreco@, twisted@, jam@cs., jkissing@ }uga.edu

Abstract

Web service technology is becoming an increasingly popular way to provide data and computational resources in the biological domain, in part because of the advanced features it provides such as language independence, platform independence, ability to discover services and ease of Web-based programmatic access. As the number of Web services provided by the bioinformatics community continues to grow, the need to use these services in combination is becoming more apparent. Typically, Web services are composed to form a process or workflow using a graphical design tool, such as a BPEL designer. However, for the scientific community, use of these BPEL tools requires a high-level of expertise. In the biological domain, there are some research efforts geared towards providing easier ways to compose biological Web services, most notably the interrelated BioMoby/Taverna projects. So far, the tools developed by such projects are still too difficult to be widely used by the average biologist. Furthermore, they tend to not fully follow Web services standards. Therefore, providing tools that lower the learning curve for Web service composition, while still complying with Web service standards could greatly benefit biologists and bioinformaticians.. In this paper, we describe WS-BioZard, a comprehensive framework for addressing this need by using wizards and semantics to provide a suite of tools that support practical semi-automatic composition of Web services in the biological domain.

1. Introduction

Over the last two decades, substantial progress has been made in the development of scientific workflows. One of the keys to make this technology useful is to have a universal infrastructure with seamless interoperability and integration capabilities. In the 1990's, the Common Object Request Broker Architecture (CORBA) advanced the state-of-the-art, but fell short in providing seamless integration and it was somewhat difficult to use [31]. The new candidate for a universal infrastructure is Web service technology. Web service technology allows operations (part of a remote program) to be invoked over the Web. Besides being universal by using XML and Internet protocols, this technology has the advantage of being easier to use and more loosely coupled than CORBA, thus allowing for easier integration.

Fundamentally, a Web service is located on the Web (has a Uniform Resource Locator (URL)) and provides one or more operations to client programs. The capabilities of a Web service are described by the Web Service Description Language (WSDL) [32] so that potential clients know how to invoke the operations and to facilitate the discovery of Web services. A service is invoked simply by sending it an XML request message and it responds with an XML response message. These XML messages are usually sealed in a Simple Object Access Protocol (SOAP) [33] envelope that puts the XML message in the body of the overall SOAP message along with additional information in a SOAP header. Note, Web services following the Representational State Transfer (REST) [34] approach eliminate the overhead of SOAP messages and send the XML messages directly.

There are several ways to discover Web services. Users may browse known sites such as XMethods (<http://www.xmethods.net/ve2/index.po>) or StrikeIron (<http://www.strikeiron.com/>) to find services. A more automatic approach is to use a Web service registry. These registries typically support the Universal Description Discovery and Integration (UDDI) standard [35].

Web service technology is beginning to be widely used in the scientific community [36]. In the biology domain, there are 1620 bioinformatics Web service registered in BioMoby service central (<http://moby.ucalgary.ca/cgi-bin/getServiceDescription>) and a bunch of service providers are listed on myGrid's website (<http://www.mygrid.org.uk/wiki/Mygrid/BiologicalWebServices>). After service discovery, the next step is to string together several Web services to accomplish a larger goal or task. This is done by composing Web services into a process where the execution of the individual Web services are coordinated by a process execution engine. With Web service composition, we come full circle back to workflows. Note, at one time, workflow also included the notion of human tasks with associated work-lists, but with the addition of WS-Human Task to Web service processes, the two technologies overlap considerably. The only major difference is that Web service processes are Web scale. Processes are defined using a process definition language such as the XML Process Definition Language (XPDL) for workflows [37] and the Business Process Execution Language (BPEL) for Web service compositions. Although Web service technology is easier to use than CORBA, unfortunately, the process language, BPEL, is more complex than XPDL (these two languages are briefly compared later in this paper).

Due to the complexity of defining processes to execute on the Web and the complexity of BPEL in particular, it is very difficult to directly write a correct BPEL specification. Since a BPEL specification is just an XML document one could attempt to do this, but it is not recommended. Today, BPEL specifications are created using Graphical User Interfaces (GUI), where BPEL constructs are dragged from a toolbar, positioned on a design canvas and wired together to form a process definition. This is much simpler than manual editing. The design tools also provide some rudimentary correctness checking capabilities. Still, to create a BPEL process to simply invoke two Web services will likely require more than ten BPEL constructs to be positioned on the canvas. Each construct will typically have several properties to be filled in. In other words, designing a BPEL process is still difficult.

This state of affairs may be appropriate for businesses which can utilize an Information Technology (IT) department, but represents an impediment for scientific communities. There are several ways to reduce the complexity of designing Web

service processes. One would be to have a simpler process definition language. Several years back, this was the case, with two of the main languages being the Web Service Flow Language (WSFL) from IBM [38] and XLANG from Microsoft [39]. The fact these two organizations worked together (with other partners) to combine these languages to form BPEL was a good thing, since having a universal infrastructure is of paramount importance. However, even if we had a simpler language, designing a process is inherently difficult.

A better course of action is to automate away some of the complexity of designing a Web service process. Automatic programming [40], where you tell the computer what you want and the computer writes the program for you, has been a goal of Computer Science that began shortly after the development of high-level programming languages. Some believe that such endeavors are folly, while others argue that it makes sense to chip away at the problem. In the field of Web services, there has been substantial and ongoing research on automatic composition of Web services. Since composing Web services can be viewed as programming in the large (the detailed code is already written in the implementations of the services), the problem is reduced to connecting the services together in a sensible manner (analogously to how a robot would move and stack blocks to achieve an overall goal). Thus, if one were able to provide unambiguous semantics for each of the operations within a Web service that includes the types of inputs and outputs as well as their preconditions and effects, one could try to use AI planning technology to solve the problem [1, 2, 3]. In simple cases, this may work. However, once the data types and control flow become complicated (e.g., with conditional branching, loops and parallel execution), the ability to create a correct and useful process degrades [4, 5]. At the present time, it seems that we need automation working alongside with human intelligence. This is the research area of semi-automatic Web service composition. This paper reports on progress made in developing a prototype tool, called WS-BioZard, that supports the semi-automatic composition of Web services¹.

The target area for our work is scientific workflows. In particular, WS-BioZard was created to support the on-the-fly composition of Web services provided by the ApiDB/EuPathDB project (<http://eupathdb.org/>) [41] as well as related services at other sites, such as the National Center for Biotechnology Information (NCBI), the European Bioinformatics Institute (EMBL-EBI), the DNA Data Bank of Japan (DDBJ) and the Protein Data Bank of Japan (PDBJ). The ApiDB project tries to provide biologists with uniform, integrated and centralized Web access to eukaryotic pathogen genomes from *Cryptosporidium*, *Giardia*, *Plasmodium*, *Toxoplasma* and *Trichomonas*. Our target audience is therefore biologists and bioinformaticians. With the advanced features provided by Web services, such as language independence, platform independence and ease of programmatic access, Web services are increasingly being used in bioinformatics to provide access to data sources as well as software to analyze the data. While individual bioinformatics Web services are useful, situations often arise where more than one service is required to perform a complete biological analysis. For example, whenever a new Genomic nucleotide sequence is annotated, biologists try to understand its functionality through searching for homologous sequences. Homologous sequences are related by evolutionary descent and are often, but not necessarily, similar at the primary amino acid sequence level. This task can be achieved through composing biological

¹

In general, when creating complex processes that involve multiple Web services, users may choose between (1) manually composing the services themselves using a GUI-based design tool, (2) using fully automatic composition software, or (3) using the hybrid approach of semi-automatic composition.

Web services: First a BLASTX² Web service could be used to identify amino acid sequences from a well-known protein sequence database. Then another Web service is used to retrieve the 'good hits' amino acid sequences. A third bioinformatics Web service is used to identify protein domains which will help to determine the functionality.

Without the help of a Web service composition tool, such tasks would be time consuming and error-prone or would require good programming skills. Compared with a traditional programming based approach, the Web service composition approach has the following benefits:

- It is easier to develop depending on the quality the design tool.
- It is easier to change a Web service during the composition through drag and drop in a design tool.
- It is easier to reuse, since the resultant process can itself be invoked as a standard Web service.

However, for biologists creating biological Web service compositions using current design tools is full of challenges. In this paper, we describe WS-BioZard, a new and comprehensive framework for lowering the complexity of Web service composition using a semi-automatic approach. WS-BioZard utilizes Semantic Annotations for WSDL (SAWSDL:www.w3.org/2002/ws/sawSDL/) and domain ontologies to add semantics to Web services. This allows us to design and implement (1) a service discovery tool with a sophisticated User Interface (UI), and (2) a service composition tool with an intuitive graphical UI and a multi-faceted UI Wizard having data mediation capabilities. The result of the composition is a BPEL process. To simplify usage further, WS-BioZard can automatically deploy the BPEL process as a standard Web service in a BPEL-compliant engine. A GUI client tool for invoking the BPEL process is also provided with WS-BioZard to invoke the BPEL process.

The remainder of this paper is structured as follows: In section 2, a brief review of existing XML-based composition languages is given. Due to the complexity of composition languages, we discuss the use of semi-automatic composition techniques to simplify the composition process from the perspective of control flow and data flow by using tools to assist the human designer. Section 3 covers related work in semi-automatic composition as well as composition techniques used in bioinformatics. Data mediation plays an important role in simplifying the creation of BPEL processes and is presented in section 4. Section 5 explains the design and implementation of WS-BioZard, which follows a layered approach. Although WS-BioZard is principally a design tool, the related issues of deployment and execution are also discussed in this section. Section 6 compares WS-BioZard with several other Web service composition tools. Finally, section 7 gives our conclusions and future work.

2. Semi-Automatic Composition

The main goal of WS-BioZard is to reduce the complexity of designing Web service processes. It is intended to be a designer only and not a process/workflow engine. Therefore, it produces design artifacts representing the composition that are fed to an execution engine. Today, these artifacts are almost always XML documents. Currently, there are five popular

²The Basic Local Alignment Search Tool (BLAST) can be used to infer gene family information and sequences' relationship, such as function or evolution, through comparing the querying sequence to sequence databases and calculating the statistical significance of matches. BLASTX is a special BLAST program to search protein databases using a translated nucleotide query.

process/workflow definition languages that are open standards: XML Process Definition Language (XPDL) [37], Web Services Business Process Execution Language (WS-BPEL) [42], Web Service Choreography Definition Language (WS-CDL) [43], Business Process Modeling Language (BPML) [44] and Unified Modeling Language (UML) [45]. Although Web service composition has benefited from experiences in the workflow domain, the standards in the two domains are distinct. The five languages fall into two categories: those that are oriented toward Web services, such as WS-BPEL and WS-CDL, and those that are more generally for business processes/workflows, including XPDL, BPML and UML.

In comparison to a workflow, Web service composition is based on interactions in the context of the Service-Oriented Architecture (SOA) [46]. Web service composition specifies an XML grammar that can be seen as a programming language to combine control and data flow with invocation of the services.

There are two main categories of Web service composition, orchestration and choreography, while the former corresponds to the traditional view of defining processes by specifying the control and data flow, the latter takes a more collaborative view and focuses on defining the message exchange [6, 7]. Although several languages have been developed, the industry has coalesced on two open standards: WS-BPEL and WS-CDL.

- As an orchestration language, OASIS standardized the Web Service Business Process Execution Language (WS-BPEL 2.0) in 2007 (based on the BPEL4WS submission co-written by IBM, Microsoft, BEA, SUN and Oracle in 2003). It specifies how Web services can be composed using process flow constructs for sequential, conditional, iterative and parallel execution, etc.
- As a choreography language, W3C standardized the Web Service Choreography Definition Language (WS-CDL) in 2004. It specifies how Web services can collaborate with each other. To join a collaboration, a Web service must follow the protocol (types and orderings of messages) specified in a WS-CDL document.

Compared with WS-CDL, WS-BPEL currently has more support from industry and wider acceptance from users. Also, there are more WS-BPEL engines (e.g., Active Endpoints' ActiveBPEL Engine, Apache ODE (Orchestration Director Engine), Sun's BPEL Service Engine, Oracle BPEL Process Manager and IBM's WebSphere Process Server) as well as graphical design tools available (e.g., Active Endpoints' ActiveBPEL Designer, Eclipse BPEL Designer, Sun's NetBeans BPEL Designer, Oracle's JDeveloper BPEL Designer and IBM's WebSphere Integration Developer).

There are other related workflow languages such as YAWL [8] and XScufl [9]. XScufl is an ongoing collaborative project between EBI, IT Innovation and the Human Genome Mapping Project (HGMP) of the Medical Research Council (MRC) and is used by Taverna, a bioinformatics tool [25]. In comparison with the above main Web service composition languages, these two have smaller tool support bases.

The choice of an SOA-oriented process definition language for WS-BioZard boils down to a trade-off between tool support and complexity. Although it is more complex, WS-BPEL's tool support is vastly greater. On the positive side, WS-BPEL is feature rich. It implements thirteen of the original twenty workflow patterns [10], including the five most basic patterns: sequence, parallel split, synchronization, exclusive choice, and simple merge. To implement these

workflow patterns, it provides many rich control flow constructs commonly associated with programming languages. Thus, we are still left with our original problem of reducing complexity, made even more critical due the complexity of BPEL. Even with a good GUI designer, creating a correct BPEL specification is very difficult for non-IT professionals. Besides the complexity of control flow and all the workflow patterns supported, data flow can also be complex. In many fields and biology in particular, there is a high level of heterogeneity in the types of data that flow between the diverse services in a BPEL process. According to [11], there are twenty different representations for just DNA sequence. BPEL allows data to be transformed in assign/copy statements using the XPath language [47]. Yet even for the IT professional, writing XPath expressions to transform the output of one service into the input of the next service is difficult and error prone (this is an actual weakness of the BPEL standard).

With manual composition using a GUI-based BPEL designer being too difficult for our target user, one could attempt to develop a tool that fully supports automatic composition. Fully automatic planners exist and their use in Web service composition has been studied in [12, 13]. The commonality between all the planning methods is that they require preconditions and effects to be specified for their correct operation. In the context planning, a precondition specifies what should be true for correct invocation of the service, and an effect specifies how the invocation of the service changes the state of the process (via its outputs and side-effects). The control flow is determined by the planner by matching the preconditions and effects to find services that can provide necessary transformations leading to a goal state. The data mediation may be provided by a planner trying to find a data transformation, which may be provided or calculated with or without annotations. The good thing about automatic composition is that since the user does not have to deal with control or data flow, the complexity of WS-BPEL can be hidden from the user.

However, there exists some downfalls to automatic composition. The problem with automatic composition is two-fold: it relies on the availability of complete formal representations of (1) the domain knowledge, and (2) the individual cases that need to be resolved, i.e., their initial state and goal state need to be formally encoded. The task of formally specifying a domain in sufficient fidelity so that it can be used for automated planning presents a huge challenge, especially in the biology domain that is advancing so rapidly that it is a moving target. Incomplete domain knowledge will often result in a situation in which an automated planner fails to produce a plan. As for specification of inputs, outputs, preconditions and effects of the Web services used in the composition, these planners rely on their semantic annotation. Unfortunately, specifying complex semantics (e.g., effects) highly accurately is in itself a challenge. Moreover, research has shown they could be good at composing sequences of services, but when we introduce the need for loops and particularly the last two basic workflow patterns (Arbitrary Cycles and Implicit Termination), they have not been shown to work well [14].

Having seen the shortfalls in the manual and fully automatic approaches, WS-BioZard implements a semi-automatic approach. While still relying on semantically annotated Web services, it overcomes the weaknesses of automatic composition because it allows users to draw upon their experience within the domain and compensate for missing or erroneous ontologies. WS-BioZard still provides automation when possible, helping the user along with the wizards, matching outputs with inputs and by reducing the number of available patterns to those deemed essential. In particular,

our approach provides nine of the original twenty patterns, including the five most basic patterns. The patterns we lose, when compared to WS-BPEL, are deferred choice, interleaved parallel routing, cancel task, and cancel case³.

We close this section by introducing the high-level architecture of WS-BioZard as well as some of the principles upon which it is based.

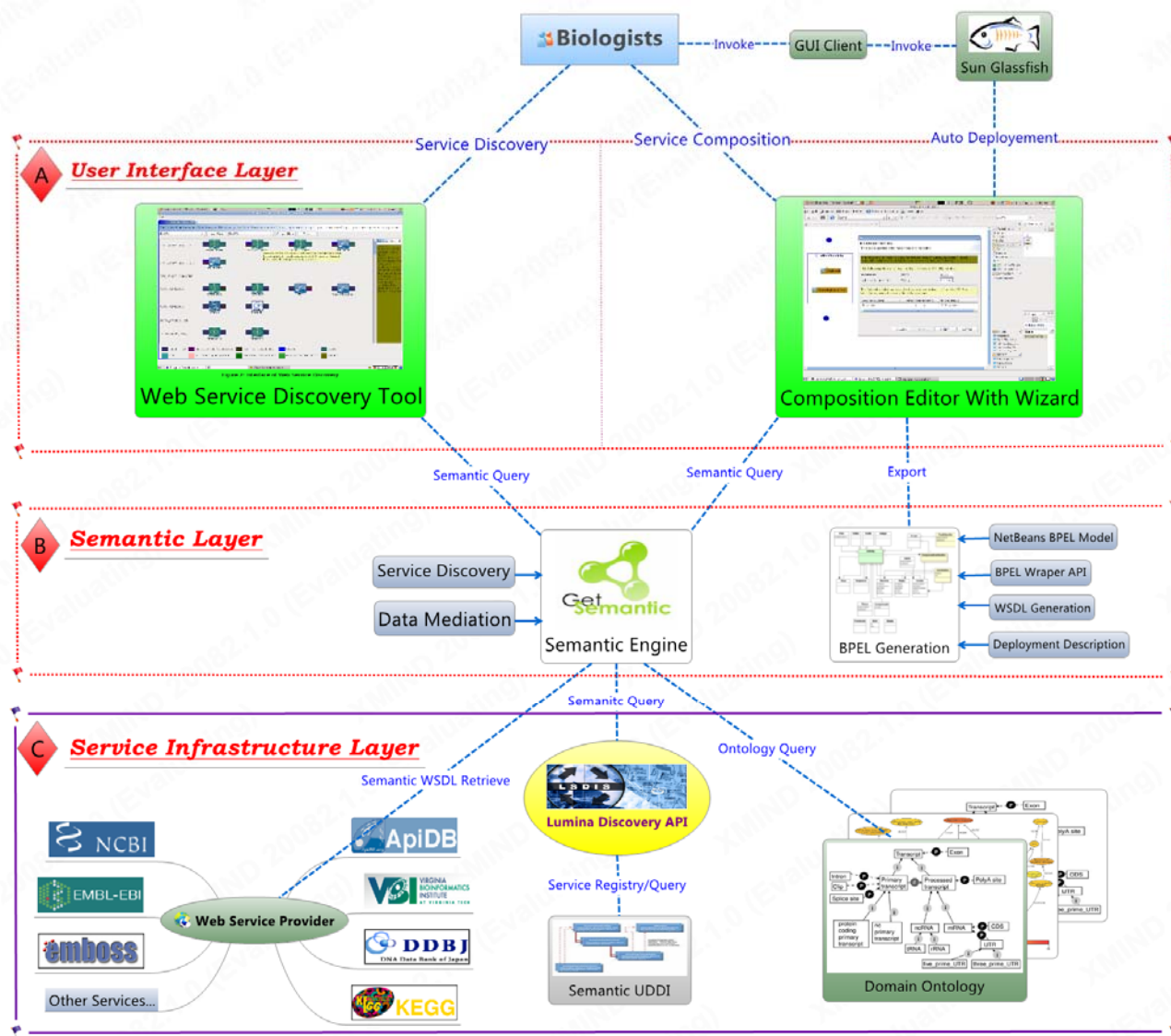


Figure 1. Architecture of WS-BioZard

WS-BioZard is architected following the layered approach shown in Figure 1 and consists of three main layers: a user interface layer, a semantic layer, a service infrastructure layer. The top layer consists of two GUI tools: a Web service discovery tool and Web service composition editor. The discovery tool is used to discover and browse the available Web

³ In our opinion, the patterns lost in our approach are of less use to our users, while we still keep much of the power of WS-BPEL with a more intuitive GUI. Thanks to the flexible architecture, we can implement those patterns when they are needed with minimum effort.

services based on criteria specified by the end user. The composition editor is used to compose Web services in a semi-automatic approach with the aid of a multi-faceted wizard. These GUI tools were developed using several basic Eclipse technologies: Standard Widget Toolkit (SWT), JFace and Graphical Editing Framework (GEF), as well as, newer and higher-level Eclipse technologies: Eclipse Modeling Framework (EMF) and Eclipse Graphical Editing Framework (GEF). The manner in which these technologies were used and the advantages they gave in the development of WS-BioZard are briefly discussed in the Appendix. The discovery and composition tools are described in detail in sections 5.1 and 5.2, respectively. Although WS-BioZard is built using the Eclipse platform, WS-BioZard can run as a stand-alone Java application outside the Eclipse workbench. The bottom layer is essentially existing technology from the METEOR-S project and includes both the Radiant (semantic annotation and publication of Web services) and Lumina (semantic discovery of Web services) software and tools. The middle layer serves as a bridge between the top layer GUI tools and bottom layer METEOR-S Semantic Web Services (SWS) technology. Section 5.3 discusses the Model Layer (middle layer), while section 5.4 discusses the Semantic Process Layer (bottom layer).

Currently, Web service technology lacks full support for service discovery, service invocation/interoperation, service negotiation and service composition. Adding semantics to Web services is the most promising solution for this problem. The objective of Semantic Web Services is to provide a knowledge representation of individual services in order to allow automatic machine processing. Adding Semantics to Web services can provide the following benefits:

- Better Discovery--Neither the Web Service Description Language (WSDL) nor the Universal Discovery and Description (UDDI) provides enough help to the client about what a Web service offers. A Semantic Web service describes its properties and capabilities which can help external applications to automatically determine its purpose and pave the path for automatic composition.
- Better Interoperability/Invocation--Semantics can help mapping the data exchanged between the services to make sure invocations operate seamlessly.
- Better Composition--Automatically check data mapping and enable data mediation.

The Semantic Annotations for WSDL and XML Schema (SAWSDL) W3C Recommendation is the first step by W3C to add semantic support to Web services technology by adding semantic annotations to WSDL components. It is now poised to unleash the power, flexibility, and logic of the next generation of Web services. Within section 5.4, we discuss how we utilize SAWSDL and how it facilitates both Web service discovery and composition for WS-BioZard.

3. Related Work

In this section, we discuss some related work in the areas of semi-automatic composition in general, as well as specific Web service composition for bioinformatics. We discuss several semi-automatic Web service composition projects using criteria such as: support for filtering out inappropriate services, suggesting partial plans, checking the validity of compositions, utilizing planning strategies, modeling environment (graphical or not), use of ontologies, control flow constructs supported, executability of compositions and compliance with standards such as BPEL.

Sirin et al. [19] present a Web service composer that uses OWL-S to provide semantics for the services. The main focus of their work is to filter out the non-compatible services at each composition step, thus helping the user to select the appropriate services. While [19] offers filtering of inappropriate web services it does not check for composition validity nor offer suggestions for partial plans, while it imposes backward chaining for the composition. On the plus side it does provide a standard output format (OWL-S) and offers a graphical modeling environment that includes the execution of the composition. Similarly, Xu et al. [20] present Dynamic Semantic Association (DSAC) which utilizes the Dynamic Semantic (DS) and Semantic Association (SA) in service matching. This work is based on SRO (an OWL-DL ontology) and focuses on utilizing a service matchmaking algorithm to provide users candidate services to select in an interactive fashion. This work supports filtering inappropriate services and achieves sequence control flow, based on a graphical user interface. However, this work cannot suggest partial plans, check the validity of compositions, or utilize a planning strategy. The composition result is not a standard and cannot be executed. Kim [21] introduced CAT (Composition Analysis Tool), a tool for interactive workflow composition. The focus of their work is to assist the user in the creation of computational workflows. The authors' work is not directly related to service composition. However, we can conceive of a computational workflow as a service composition. The activities of the workflow compare to service operations that realize data transformations. Its strength is in evaluating "well-formedness" for the composition and suggesting fixes, but does not provide for filtering nor control constructs and their modeling environment is textual. Hakimpour [22] introduced Internet Reasoning Service (IRS), a Semantic Web Services framework. One of their implementations, IRS-III, includes a tool that supports a user-guided interactive composition approach by recommending component Web services according to the composition context. Their approach uses Web Services Modeling Ontology (WSMO) as the language to semantically describe the functionality of the Web services. IRS-III's main strength is in that it provides an execution engine, but does not provide for help in the composition process by filtering or suggesting partial plans.

The two main projects focusing on bioinformatics Web services composition are myGrid and BioMoby. The myGrid project is part of the UK government's e-Science program [23], and aims to provide a comprehensive middleware suite specifically to support data intensive in-silico experiments in biology. The core components are based on Web service technology, which can be adopted in a "pick and mix" way for developers and tool builders to form, execute, manage and share discovery experiments. As a subproject of myGrid, Taverna is a bioinformatics Web service composition editor using the XScufl language with a friendly GUI. Compared with a general BPEL editor, it is much easier and more intuitive to use. However, it has two main drawbacks. First, the workflow generated by it can not be run outside of Taverna as a standard Web service, which limits its functionality and usability. Second, the user usually needs to manually handle data mapping using a scripting language such as the Java-based Bean Shell language. This feature provides high flexibility with the cost of making Taverna hard to use by the average biologist.

BioMoby [24] is an open source research project which implements an architecture for the discovery and distribution of biological data through Web services; data and services are decentralized, but the availability of these resources, and the instructions for interacting with them, are registered in a central location called MOBY Central. MOBY Central provides an object-driven registry query system with its own object and service ontologies. The strength of this initiative is in the service registration, discovery and transaction process with semantic information. However, BioMOBY by itself is limited

in scope, in that it does not include facilities for creating Web service processes or workflows. In order to compose BioMoby services, one can use the BioMoby plug-in to Taverna [26] and then use the graphical workflow designer provided with Taverna (an example of the use of this designer is given in section 6).

4. Data Mediation

As processes consist of control and data flow, these become two avenues with which to attack the problem of complexity. We have discussed in section 3 how WS-BioZard reduces complexity by limiting the process flow constructs to the essential ones. Furthermore, some of the BPEL elements need not be explicitly placed in the process by the designer, as WS-BioZard inserts them where needed. In section 6, we will show how this results in a much smaller and simpler design diagram. WS-BioZard reduces the complexity of designing BPEL processes by introducing mediators and wizards for automating the design: either fully automatically in the case of mediators or semi-automatically in the case of wizards. In this section, we focus on the techniques used for data mediation. A more detailed description of the components making up WS-BioZard is given in section 5.

Data mediation may be the most challenging task in Web service composition. It is even worse in the bioinformatics Web service composition domain, considering the lack of a universal naming convention, extreme diversity of terms and rich data schema. Manual data mediation could be quite frustrating since one needs to map the outputs of one service to the inputs of another using, for example, XPath within BPEL assign/copy constructs. Assistance with this could pay large dividends. In WS-BioZard, we developed an algorithm that utilizes bottom-up SAWSDL annotations to provide a practical automatic data mediation solution. This complements the data mediation approach based on top-down SAWSDL annotations presented in [17].

Data mediation is not a new issue. It has been well researched since the inception of federated databases [28, 29]. There are three layers in schema/data heterogeneity: structural, syntactic and semantic. Compared with structural and syntactic heterogeneity, semantic heterogeneity is more difficult and complex. Semantic mappings mean translating data from one data source into another, while preserving the semantics of the data [27]. Four levels of heterogeneity are listed in [17]:

- Syntactic heterogeneity: differences in the languages used for representing elements.
- Model/Representational heterogeneity: differences in the underlying models or their representations.
- Semantic heterogeneity:
 - Same name, different meaning: for example the term ID can mean either student ID or class ID.
 - Same meaning, different name: for example, lastname and familyname have the same meaning.
- Structural heterogeneity: different structures in elements. For example: company(CEOname, CEOemail, companyName) vs. company(CEO(name, email), companyName) are two elements with different structures.

Web services communicate by exchanging XML messages with each other. The types of the messages are determined by combining the WSDL message parts with any XSD schema definitions (WSDL 2.0 eliminates message parts, so types are wholly defined using XSD). One may view these types as a tree. In effect, SAWSDL allows the nodes of these type

trees to be annotated (more about this in section 5.3). Two Web services can communicate if they annotate their WSDL specifications with the same or alignable ontologies. For example, if a Web service annotates the output of its Web service with concept C (typically specified in OWL) of a given ontology and another service takes this concept as input, these services can communicate. Note, this does not require the two services to use the same types, only that their types are mappable to concept C. In this way, SAWSDL allows for, yet handles heterogeneity.

There are two ways to use annotations to establish mappings between Web services, allowing the output of one service to be mapped to the input of the next service: the top-down matching approach and the bottom-up matching approach.

4.1. The Top-down matching approach

As the name implies, this approach matches the top elements of the output and input messages. Even though the top elements conceptually match, their XSD type trees can have differences at several levels. Hence, for matching from the top to the bottom elements, there are various types of heterogeneity to be considered. The prototypes developed in [17] and [48] both use this top-down approach for their data mediation. To handle heterogeneity top-down, they use the lifting/lowering schema mapping method. They first transform (lifting schema mapping) the output message of the first Web service an OWL instance of concept C. Next, they transform (lowering schema mapping) this intermediate OWL instance to the appropriate type for the input message of the second Web service. Of course, one may wish to optimize out the intermediate OWL instance by composing to the two transformations. Multiple languages can be used for specifying the transformations, for example, XSLT and XQuery have been tested in this role. They provide concise transformations, but require some level of sophistication to create. Another complication of the top-down approach is that these transformations must be applied to XML messages as the process/workflow runs. There are several ways to implement this: add them as additional steps in a SOAP Engines (e.g., Axis2) pipeline ([17] experimented with this), extend BPEL to permit XSLT transformation in `<assign>/<copy>` elements, or use a proxy Web service. None of these approaches is particularly easy, but in the long run one of them may become the preferred solution. The following example shows the top-down approach:

Suppose the complexType `Company(CEOname, CEOemail, companyName)` is the output of the first web service and the complexType `company(CEO(name, email), name)` is the input of the second Web service. Data mediation is used to convert the output message of the first Web service to the input message of the second Web service. The top-down approach will match and map the top elements “company” with “company”. Because the structure and attributes of “company” are different in the two messages, XSLT will be used to transform the nodes to ontological concept or properties (lifting) and then fill each node of the target message schema tree (lowering) as shown in Figure 2.

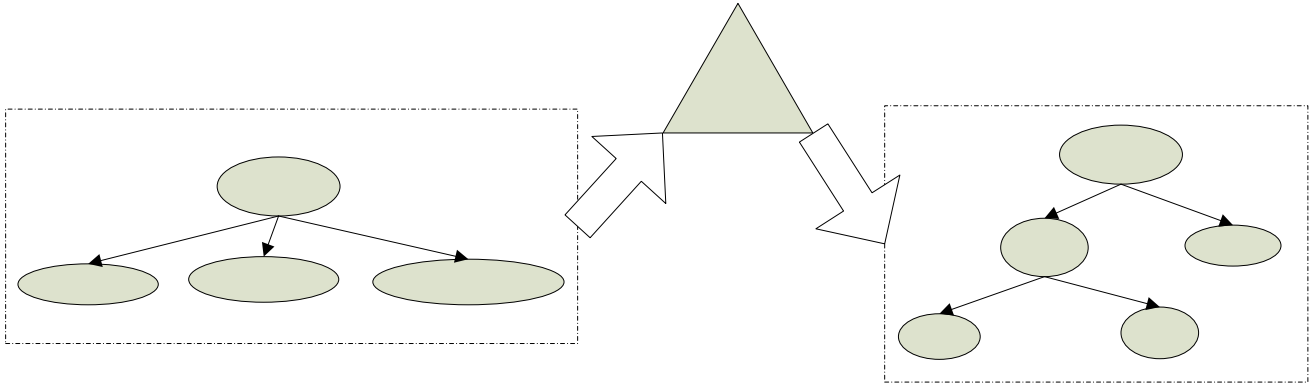


Figure 2. Example of Top-Down Approach

4.2 The Bottom-up matching approach

The SAWSDL standard is flexible in how it allows data types to be annotated. One may specify how whole data types (e.g., an XSD complexType) can be transformed by placing the mapping at the root of the tree as in the top-down approach, or one may specify how to map the leaves of the tree as in the bottom-up approach. Assuming the whole is more than the sum of its parts, semantics may be lost in the bottom-up approach. On the other hand, it has the advantage that the mappings are simpler and in common cases should be effective. Keeping with the theme of avoiding complexity, WS-BioZard's data mediation subsystem uses the bottom-up approach.

For common cases, XPath can be used to map the bottom-level elements. These XPath expressions can then be placed in the <assign>/<copy> statements in BPEL. It is hard to imagine a simpler solution. In complex cases this may fail and we intend to test its limits as we evaluate WS-BioZard as it is put into practice. Conceptually, the lifting mapping goes into the <from> part of <copy>, while the lowering mapping goes into the <to> part (see section 5.3 for an example). These XPath expressions are automatically generated at design time from the annotated SAWDL files for the two communicating Web services. SAWDL model references establish links from the XSD type tree to the ontology. The goal is simply to provide all the necessary inputs to the second Web service. Thus, for each leaf of this type tree, an XPath is generated. The model reference to the construct in the ontology is then traced back to find the conceptually matching leaf in the XSD type tree of the first services output. If the connection is made, the XPath for this part is generated. The first XPath expression is placed in the BPEL <from>, and the second XPath expression is placed in the BPEL <to>.

Algorithmically, at run time the following transformations are performed on the data being exchanged by two Web services:

$$X_i = \text{XPath}_{\text{lower-}i}(\text{XPath}_{\text{lift-o}}(X_o))$$

where X_o is the XML output message produced by the first Web service and X_i is the XML input message to be sent to the second Web service. In particular, for each leaf element in X_i , a path expression is created to lift the output and then lower it the input. Generating these XPath expressions is, however, a bit more complex. We need to handle three cases. Suppose

Lifting

CEOname CEOemail companyName

the first Web service's output annotates to class C_o in the ontology, and the second service's input annotates to class C_i (annotation using properties is also supported). In the first case, C_i is the same, an equivalent or a super-class of C_o , so all the input values are available from the output. In the second case, C_i and C_o have overlapping properties, so some, but not all values are available from the output. Finally, in the third case, C_i and C_o have no properties in common (unlikely to happen in practice assuming appropriate service discovery).

```

if  $C_i = C_o$  or  $C_i \equiv C_o$  or  $C_i \in \text{superclass}(C_o)$  // case 1
  XPathlift-i = path expressions generated from first services SAWSDL
  XPathlower-i = path expressions generated from second services SAWSDL
elseif  $C_i \sqcap C_o \neq \phi$  // case 2
  XPathlift-i = path expressions generated from first services SAWSDL
  XPathlower-i = path expressions created using the data mediation wizard
else // case 3
  Reject WS2
endif

```

For case 1, WS-BioZard will automatically generate the BPEL <assign>/<copy> elements. For case 2, the data mediation wizard will popup and assist the user to produce expressions for missing values. WS-BioZard ability to automatically generate XPath expressions greatly simplifies the composition task for the user.

Let us again consider the example given in section 4.1. If we use the bottom-up approach, we will parse both of the messages' XSD types. The bottom elements for the output of the first Web service are {CEOname, CEOemail, companyName} and their annotations are {humanName, email, companyName}, respectively. The bottom elements for the input of the second web service are {name, email, name} and their annotations are {humanName, email, companyName}, respectively. Therefore, we can as shown in Figure 3 match/map the elements as follows: CEOname—name, CEOemail—email, companyName—name.

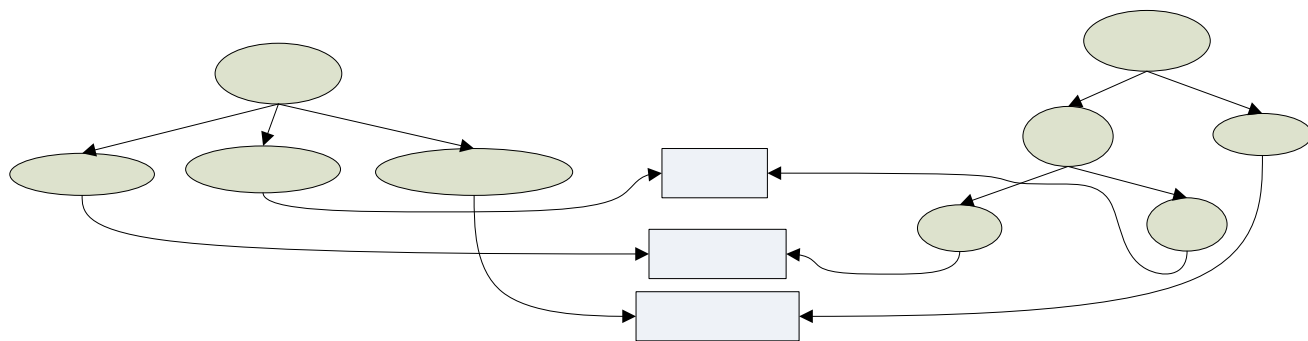


Figure 3. Example of Bottom-Up Approach

The implementation details of how this data mediation works are discussed in section 5.3.

5. Design and Implementation of WS-BioZard

In this section, we discuss the four main components of WS-BioZard: the Web Service Discovery Tool, the Web Service Composition Editor (both in the top layer), the Semantic layer (middle layer) and the Service Infrastructure layer (bottom layer). We close by discussing deployment and execution of BPEL processes.

5.1. Web Service Discovery Tool

Through the intuitive Web service discovery interface shown in Figure 4, biologists can view basic Web service information, such as service name, provider represented by the central icon, description, input/output data type and functionality. Furthermore, they can query/filter Web services by input or output data type. With the aid of this tool, biologists can get the overall information about all the available Web services, which will be helpful to the next step—Web service composition. There are five main regions in the tool: The main central region contains all the discovered services. The left column consists of top-level ontological service functionalities, while the right column gives the detailed description of the selected Web service. The bottom region provides a color coded key for the services' input/output data types, while the top region is used filtering of/querying for Web services based on, for example, input/output data types.

As shown in Figure 4, each service icon is composed of the service provider's logo (in the middle of the icon), the service name (at the bottom of the icon) and input/output data types represented by color coded tab and attached to the middle icon. Each color represents a type corresponding to the bottom area, and the same color shared by two services' output and input means that the two services could be composed together (i.e., there is at least a partial match). In Figure 4, there are ten input/output data types: identifier, nucleotide_sequence, peptide_sequence, fasta, blast, gff, multiple_alignment, protein_structure, phylogenetic_tree and other. The input/output data type information is stored in a domain ontology and referenced by SAWSDL annotations in their XML schema part. The ontology as well as the chosen ontological functionalities and types are customizable by the user. Whenever the mouse hovers on a service, the description of this service will popup. When a service is selected, the detailed description of this service is shown in the right column. At the same time, all the incompatible services⁴ are automatically filtered out. Furthermore, biologists can query/filter out Web service by input/output data type using features given in the top region.

The services are dynamically retrieved based on queries to a semantic UDDI registry, which contains all the registered WSDLs with SAWSDL annotations. The services are arranged by rows, with each row representing a functional category (or top-level functional concept) from the given ontology. Currently, there are five categories: “similarity_searches”, “sequence_convert”, “data_retrieval”, “gene_prediction” and “phylogeny_tree”. Whenever there is a need to extend with a new functional category, the only place to be modified is the ontology. Once the new functionality category is properly

⁴The input of these services are not compatible/matched with the output of the selected one. In other words, the incompatible services can not be composed with the selected service because the input and output can not match.

inserted into the ontology, it can be automatically shown and used by the service discovery tool⁵. Any service annotated with this new functionality will be shown in the Web service discovery interface automatically, as long as it is registered with a semantic UDDI.

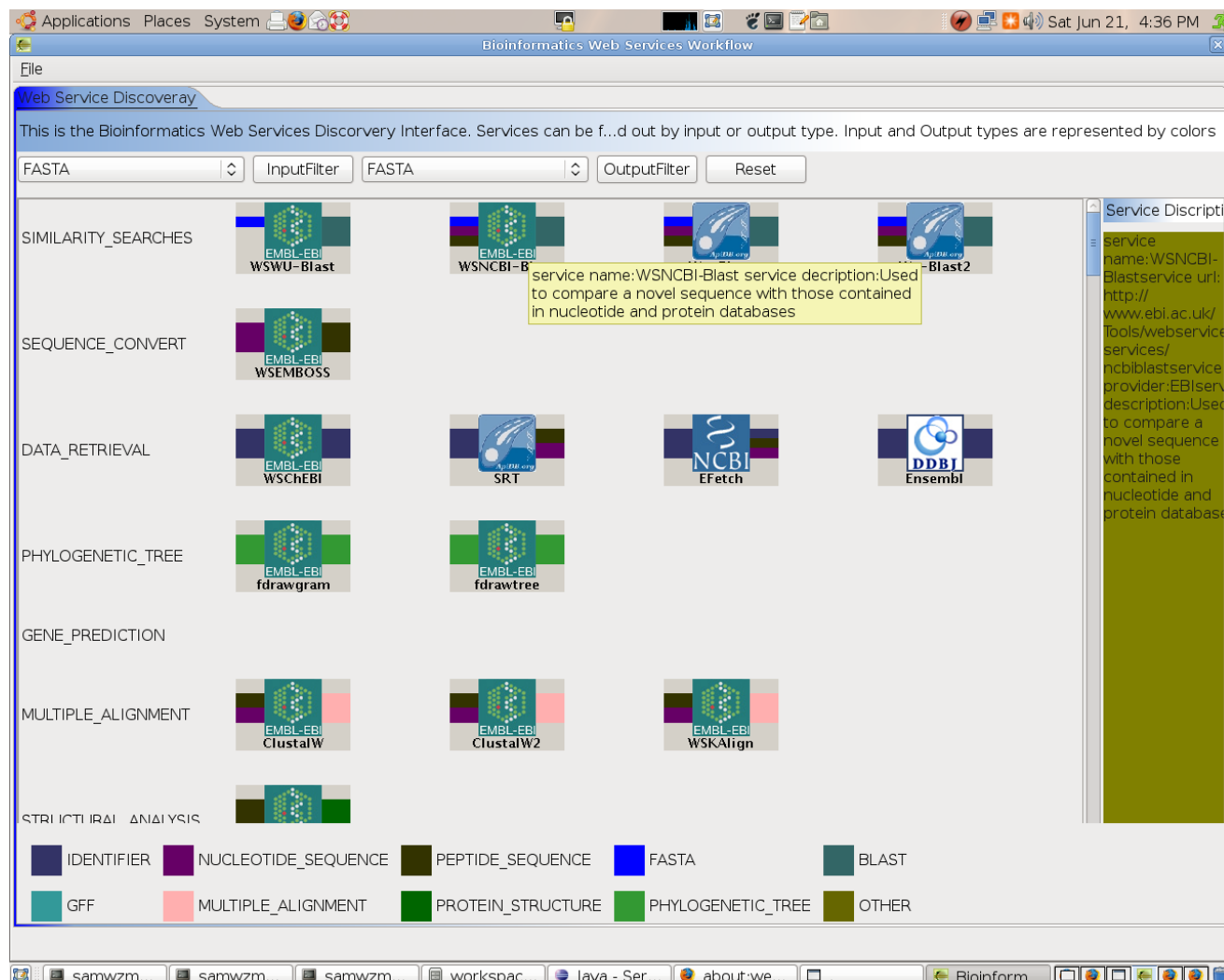


Figure 4. Web Service Discovery Tool

5.2. Web Service Composition Editor

In this subsection, we discuss how the Composition Editor utilizes a multi-faceted wizard to assist biologists in composing Web services. We then discuss how our editor simplifies the service composition process.

5.2.1. A multi-faceted wizard. A wizard is a user interface element designed to assist end users with a specific task. A well-defined wizard should hide much of the complexity involved in the task from the user. At the same time, it should

⁵In the future, the service discovery tool will be more configurable. The advanced user can choose ontology and select functional categories and data

provide a supplementary rather than substitutive way to accomplish the task to avoid restricting functionality for advanced users [15]. The end user is led through a sequence of dialogs to collect all the necessary information and user input for the task. A general purpose wizard may be too complex and difficult to implement. However, a wizard for a specific domain can be very useful and can lower a user's burden, especially when performing a complex task [16]. Restricting the wizard to specific domains within bioinformatics allows the wizard to utilize one or a few related domain ontologies (i.e., complex and risky alignment of diverse ontologies can be avoided).

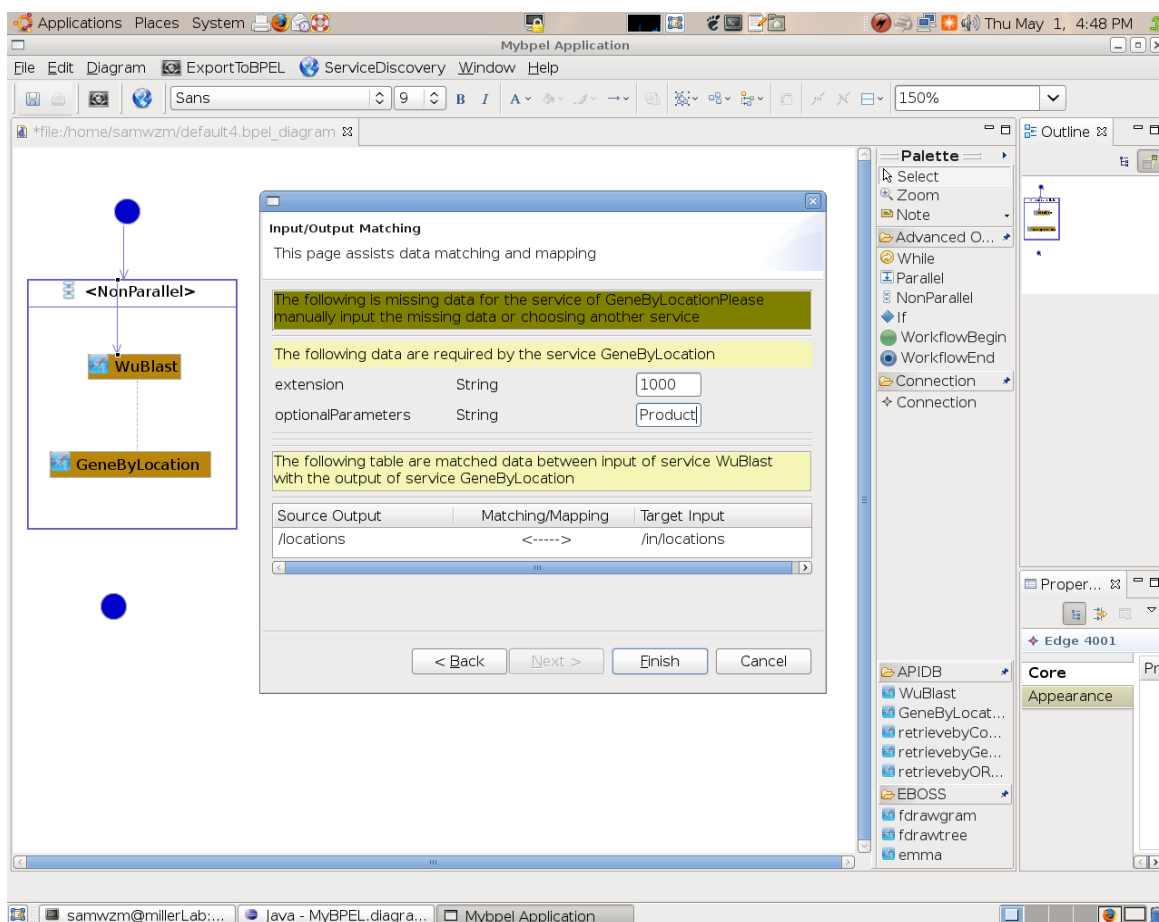


Figure 5. Our BPEL Editor with Wizard

In our framework, the role of the wizard is to be a biologist's assistant. For example, a wizard can automatically pop up when a biologist tries to connect any two services, providing information about service input, output and functionality. Furthermore, a wizard can be used to check the services mapping/matching as well as assist in data mediation. Because of the above benefits provided by our wizard, we believe that our architecture significantly reduces the complexity of bioinformatics Web services composition. Figure 5 shows the wizard assisting in data mapping/matching in our Web service composition editor.

types to customize the service discovery tool.

5.2.2. Simplifying service composition. Since BPEL is a very general orchestration language, most BPEL editors will reflect this through an interface with all 13 BPEL constructs/workflow patterns. This makes the editor too complex to be willingly used by the average biologist. In our composition editor, multiple approaches are used to lower the learning curve for biologists.

First, we make the palette more intuitive by separating the Web service selection from the flow control activities and hide most of the BPEL elements from the end user. For example, there are no “receive”, “reply”, “partnerLink”, “assign” or “invoke” in the BPEL editor canvas. There are two reasons why we choose this approach:

- End users can focus more on business logic, instead of detailed (and often subtle) issues of process flow.
- Those elements have clear meaning for computer science users. However, they may not be easy to understand by biologists.

This simplification for end users (biologists) comes with the cost of implementation complexity for us (the developers). For general BPEL editors such as ActiveBPEL, NetBeans BPEL and Eclipse BPEL, each BPEL element shown in the editor canvas corresponds to its implementation in the BPEL model. Whenever an end user modifies a BPEL process (either insert or delete), it needs less work to keep synchronization between the BPEL process shown in the canvas with the BPEL model. However, in our approach, we must implement a bridge between the BPEL process shown with the underlying full BPEL model. This bridge is used to keep synchronization between our BPEL editor with the BPEL model. For example, with WS-BioZard, when a user drags-and-drops a service into the editor, a namespace, input/output variables, partnerlink, assign and invoke will be automatically generated (see section 5.3.3 for details). Second, whenever there is a connection between two services, a wizard will popup and collect information from a user to create BPEL activities, such as “assign”, “copy” and “invoke”. Whenever the connection is reoriented or deleted, the corresponding actions will be taken to make the correct change. All these activities are transparent to the end user. Third, each possible Web service selection actually represents a service operation, for the following three reasons:

- Eventually an operation will be chosen to invoke the service.
- Many Web services include multiple operations and the end user is often more interested in a specific operation rather than the overall Web service itself.
- Automatic data matching/mapping can only be achieved after the operation is finalized

5.3. The Semantic Layer

The top layer consists primarily of Graphical User Interface code, while the bottom layer primarily supports the registration and discovery of semantic Web services. The logic and algorithms supporting semi-automatic composition belong to this layer. It also includes code for utilizing software in the infrastructure as well as code for serializing the designs and generating design artifacts (BPEL and process WSDL). Since SAWSDL plays a central role in this layer, we begin by presenting it in more detail. We then discuss the components making up this middle layer.

The Semantic Annotations for WSDL and XML Schema (SAWSDL) W3C Recommendation is the first step by W3C to add semantic support to Web service technology by adding semantic annotations to WSDL components. It is now poised

to unleash the power, flexibility, and logic of the next generation of Web services. One of the key design principles of SAWSDL is that it enables semantic annotations for Web services using and building on the existing extensibility framework of WSDL. This is the fundamental feature for our framework, since it is incremental (i.e., it will function in a non-semantic world, yet work better as semantics are added).

SAWSDL defines the following three new extensibility attributes for WSDL 2.0 (WSDL 1.1 is also supported) elements to enable semantic annotation of WSDL components:

- An extension attribute, named `modelReference`, to specify the association between a WSDL component and a concept in some semantic model. This `modelReference` attribute can be used to annotate XML Schema type definitions, element declarations, and attribute declarations as well as WSDL interfaces, operations, and faults. In our framework, we use `modelReference` to annotate service operations and XML schema type definitions.
- Two extension attributes, named `liftingSchemaMapping` and `loweringSchemaMapping`, that are added to XML Schema element declarations and type definitions for specifying mappings between semantic data and XML. These mappings can be used during service invocation. (from W3C website). In our framework, we use them to aid data mediation as well.

SAWSDL is easy to learn and use as it extends WSDL. This represents an evolutionary and compatible upgrade of existing Web services standards. Currently, there are already several open-source tools for SAWSDL which are used in our framework, including: SAWSDL4J is an object model for SAWSDL documents. Radiant is a UI for annotating existing WSDL documents based on WSDL-S or SAWSDL via, for example, an OWL Ontology. Lumina is a Semantic Web Service discovery tool that complements Radiant.

There are several components in this Semantic Layer. These components are utilized by both the composition designer (section 5.1) and the discovery tool (section 5.2). A graphical representation is used to save the composition designer's results to disk, which can be done automatically by Eclipse GMF. BPEL serialization uses the NetBeans BPEL API to generate BPEL. The model part of NetBeans SOA project is used as our model implementation. It is an open source project and implements all the features of WS-BPEL.

An important contribution in this project is the design and implementation of the semantic engine. It handles the semantic requests from the UI module, providing information to assist service discovery and data mapping. It connects with the Semantic Web service registry and other components of the Service Infrastructure layer to accomplish its function. Currently, commercial Web service technology lacks full support for service discovery, service invocation/interoperation, service negotiation and service composition. Adding semantics to Web services is the most promising solution for this problem as discussed in section 4. Because of the flexible design of WS-BioZard, it has the potential capability to switch to other semantic approach. The only modification will be the semantic engine to adopt this change.

Since there are a large number of biological Web services available online, locating the relevant ones manually would be time consuming. Being able to find the most relevant service based on input, output and functionality automatically is very useful. The Lumina Discovery API (see section 5.3) is used to discover Semantic Web Services based upon

SAWSDL annotations. The algorithm can find the most relevant Web service through calculating semantic Web service property similarity, based on concepts, semantic relations, and their common and distinguishing features. A Web service's input, output, and functionality are considered during the similarity calculation.

5.3.1. Data Mediation Implementation Issues

Once the end user is ready to compose Web services, we have developed an algorithm to find out if two Web services match or not (refer to section 4.2). If the match is close enough, a means for providing missing information can be given manually by the end user at design-time (see the wizard interface in Figure 3). The following example shows this process (please refer this example to section 6). Table 4 shows the data schema annotated using SAWSDL:

<p>WuBlast Output Schema</p>	<pre><xsd:element name="wuBlastResponse"> <xsd:complexType> <xsd:sequence> <xsd:element name="locations" type="xsd:string" sawsdl:modelReference="http://purl.org/obo/owl/sequence#location_text" /> </xsd:sequence> </xsd:complexType> </xsd:element></pre>
<p>GeneByLocation Input Schema</p>	<pre><xsd:complexType name="inputType"> <xsd:sequence> <xsd:element name="locations" type="xsd:string" sawsdl:modelReference="http://purl.org/obo/owl/sequence#location_text"></xsd:element> <xsd:element name="optionalParameters" type="xsd:string" sawsdl:modelReference="http://purl.org/obo/owl/sequence#OPTIONAL"></xsd:element> <xsd:element name="extension" type="xsd:int" sawsdl:modelReference="http://purl.org/obo/owl/sequence#extent"></xsd:element> </xsd:sequence> </xsd:complexType></pre>

Table 4. Data Schema Annotated by SAWSDL

Suppose the user inputs “0” for extension and “Product” for “optionalParameters”, the following <assign> activity will be generated by WS-BioZard:

```
<assign name="invoke102Assign">
  <copy><from><literal>Product</literal></from>
  <to>$invoke103inputVar.parameters/ns102:in/ns102:optionalParameters</to></copy>

  <copy><from><literal>1000</literal></from>
  <to>$invoke103inputVar.parameters/ns102:in/ns102:extension</to></copy>

  <copy><from>$wuBlast102outputVar.parameters/ns101:locations</from>
  <to>$invoke103inputVar.parameters/ns102:in/ns102:locations</to></copy>
</assign>
```

5.3.2. BPEL Generation

Considering the complexity of the WS-BPEL specification and the availability of free open-source BPEL implementations, making our own BPEL model is not a good choice. Currently, there are two free open-source BPEL implementations, NetBeans BPEL and Eclipse BPEL. Both are mature and conform to WS-BPEL 2.0⁶. Although NetBeans BPEL is meant for their own IDE and very hard to separate from their IDE codebase, NetBeans provides a nice editor with automatic execution engine deployment. This feature greatly simplifies BPEL process validation both at design time and run time, which is the main reason why we choose NetBeans BPEL as our BPEL model implementation.

Based on the NetBeans BPEL model, we made a wrapper API corresponding to “BPEL Wrapper API” module, which is a bridge between our BPEL editor and the NetBeans BPEL model. Since we simplify of our BPEL editor (hiding “receive”, “reply”, “partnerLink”, “invoke”, “assign” from the end user), we must implement a wrapper/bridge to keep synchronization between the BPEL process and the BPEL model. WS-BioZard also simplifies BPEL deployment via the automatic generation of the process WSDL files as well as NetBeans generated deployment descriptors.

5.4. Service Infrastructure Layer

The semantic layer described in section 5.3 relies on the semantics provided by the bottom layer, the Service Infrastructure layer. There are two main components in this layer: Radiant and Lumina. These tools were initially developed as part of the METEOR-S project [30]. During the standardization process of SAWSDL, they were updated and made generally available on the W3C SAWSDL Website. Radiant is a GUI tool that allows users to rapidly annotate WDSL documents with semantic annotation (model references to ontological concepts) and then publish the annotated services in a semantic registry. Lumina is a tool for discovering semantic Web services and is divided into a GUI part and a discovery API that allows programs to find Web services stored in a semantically enhanced UDDI registry. The discovery API of Lumina is being reused by our discovery tool described in section 5.1.

Although Lumina can work on regular UDDI registries, improved discovery is provided when semantic annotations are available. Therefore, it is ideal if service providers go to the extra effort to annotate their services following the SAWSDL standard. This procedure provides the key support for service discovery and data mapping. Multiple tools can achieve this task including Radiant and WSMO Studio. Radiant is an open source Eclipse plug-in project developed in the LSDIS lab at the University of Georgia. Since Radiant can not only annotate, but also publish the annotated service to a UDDI registry with the assistance of the discovery API, it is used in our framework.

Lumina's Discovery API [18] provides a semantic layer for Web services on top of a traditional UDDI registry. The API provides semantic equivalents to two of the most important operations in UDDI, publishing and discovering. Publishing is done with Web services that were annotated according to the SAWSDL standard storing all annotations present in the SAWSDL document. The API's discovery mechanism works against an ontology and is able to provide services that are compatible based on reasoning over the ontology.

⁶Although both of them conforms to WS-BPEL 2.0, their implementation is slightly different. Hence, not all the BPEL process generated by Eclipse

Ontology plays a fundamental part for semantics by providing the formal definition and relationship between concepts. Because of the complexity of the biological domain, diversity of biological terms and lack of an universal naming convention, ontology is particularly important in the biological domain. For testing our prototype implementation of WS-BioZard, we extended the well-known Sequence Ontology (SO) (www.sequenceontology.org/) as our domain ontology. A joint effort by genome annotation centers, SO includes a set of terms and relationships used to describe the features and attributes of biological sequences. However, our framework can support multiple domain ontologies to provide broader semantics.

5.5. Deploying and Executing BPEL Processes

Once the completed BPEL specification is created, one needs to deploy it and then execute the BPEL process. Deployment installs the BPEL process on a server hosting a BPEL execution engine. The BPEL engine will treat the BPEL specification like a high-level scripting language, one oriented toward processing XML and invoking services. We have tested our WS-BioZard prototype primarily using Sun's BPEL Server Engine. While many BPEL engines require a deployment descriptor, Sun's BPEL does not, so it simplified the testing of our prototype. It allows automatic deployment to its Glassfish application server.

Once the BPEL specification is deployed, a variety of clients can run the BPEL process. Since the overall BPEL process itself is a regular Web service, it can be invoked by any consumer. Such clients may take the form of a client program or a GUI based Web service invocation tool such as SoapUI (www.soapui.org). To simplify such invocations, we have made a simple Java Applet to invoke such Web services from a Web browser (such client tools could also be developed using JavaScript).

As a standard Web service, the overall BPEL process must have a WSDL specification to represent it. Currently, none of BPEL editors can automatically generate this WSDL from the BPEL process specification, including the ones we have tried: Active Endpoints' ActiveBPEL Designer, Eclipse BPEL Designer, Sun's NetBeans BPEL Designer, Oracle's JDeveloper BPEL Designer. Considering the complexity of WSDL specifications and their associated XML schema, manually creating WSDL is challenging for computer scientists, let alone for biologists. However, without this WSDL definition, no BPEL process can be deployed and invoked by any consumer. Since our goal is to substantially reduce the complexity of designing, deploying and executing BPEL processes, WS-BioZard includes the capability to automatically generate WSDL specifications from the BPEL process specification⁷. It parses the BPEL and creates a WSDL operation specification from the information found in the BPEL receive and reply elements as well as variable and namespace specifications.

6. Comparison of Web Service Composition Design Tools

BPEL can be run under NetBeans BPEL.

⁷As long as the first BPEL activity is "receive" with "instance=yes" and the last BPEL activity is "reply", WS-BioZard can handle it correctly. This covers most of cases.

In this section, we present a bioinformatics task that requires the composition of two Web services provided by ApiDB: WuBlast and GeneByLocation. The two Web services have been written, semantically annotated and published. The purpose of this example is to compare our system with Taverna, Active-BPEL and NetBeans in facilitating the composition task for the user.

The first service "WuBlast" runs the BLAST algorithm developed at Washington University (<http://blast.wustl.edu/>). BLAST can be used to infer gene family information and sequence relationships, such as function or evolution, through comparing the querying sequence to sequence databases and calculating the statistical significance of matches. A BLAST output is a set of matches (or "hits") to well-known sequences from a database. These matches are sequences defined by a start and stop coordinate values in the genome sequence and are ordered by a confidence score. The matches provide information about the "query sequence": important functional information and evolutionary relationships between the sequences. There is a lot of information buried in these hits that might be more useful to biologists. For example, the user might want to find genes in these matching sequences. For this, we need a second web service "genesByLocation" that tells us, given the hit location (start, stop), if there are any genes in it.

Composing these two services to form a single process can significantly improve efficiency. The biologists no longer will need to save the results of the first step, convert it to the proper format and then carry out the second step. Furthermore, the process could have been designed with a loop so that many such input sequences could be handled at once. To compose these two services, first we need to filter the BLAST results according to some well-defined criteria, such as percentages that measure how identical the matches are, and how well the query sequence is covered by the matches. For each hit, the second Web service will identify all the genes in it. The gene sequence might be fully inserted in the hit, or it might fall partly within it; the user will input a range or extension, a number of nucleotides, which will determine how far beyond the hit start and stop coordinates to search for annotated genes.

Figure 6 shows the BPEL process designed using WS-BioZard. The center part is the design canvas on which users can drag and drop components and visually create the BPEL process. On the top, all the buttons of the toolbar can customize the look of the BPEL process graph such as changing the size, color, font and automatically aligning all the components in the graph. The tools on the top right include zoom, adding notes, etc. Beside these tools is the thumbnail of the BPEL process graph. Below the tools section are all the containers which the user can use in BPEL process, such as while, if, etc. They can be dragged into the canvas to create structures for the BPEL process and then add Web services into these containers. All the available Web services are on the bottom right, so user can drag and drop the Web service icon to visually compose the BPEL process. When the user clicks any component of the BPEL process on the canvas, the properties of the component will show up in the property section on the bottom right.

To create the BPEL process for the sample workflow we described above, we should drag the "WorkflowBegin" into the canvas. After that, we should add one sequence container and two Web services (wuBlast, geneByLocation) into it. After we add the "WorkflowEnd", we can use the "connection" to connect all components together. As soon as we connect the two Web services together, a wizard window will popup (see Figure 6) for user to enter values or expressions for the non-

matched inputs of the second Web service. After we finish composing the BPEL process, our tool can automatically deploy the BPEL process to Sun's Glassfish Application Server.

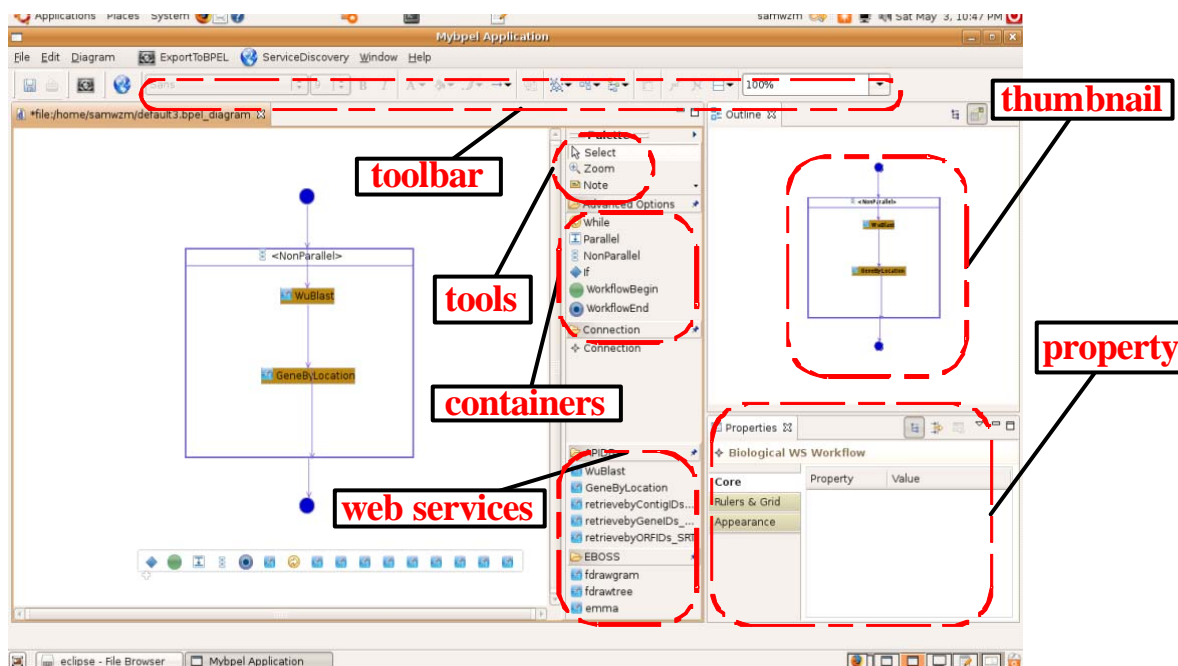


Figure 6. Workflow Made by Our BPEL Editor

Without the help of a service composition tool, the above task would be very time consuming, not only due to the number of service calls required for even a small number of hits, but also considering that one might want to explore different search databases, different filtering criteria and different range values. The following three subsections compare WS-BioZard with three popular alternatives means for composing the same sample workflow: ActiveBPEL Designer (6.1), Sun's NetBeans BPEL Designer (6.2) and Taverna's Designer (6.3).

6.1. ActiveBPEL Designer

The ActiveBPEL Designer, developed by Active Endpoints, Inc, is a visual environment for designing, developing, testing and deploying WS-BPEL 2.0 (BPEL) compliant process definitions. As shown in Figure 7, we use the

ActiveBPEL Designer 4.1 to compose the sample bioinformatics BPEL process described above. Compared to WS-BioZard, the ActiveBPEL designer provides less automation and is harder for non-computer science users to use.

Automation:

- A BPEL process itself is deployed as Web service, so a WSDL file has to be created to represent web service of BPEL process itself. Normally it is hard for biologists to create such WSDL file. Our WS-BioZard will automatically create the WSDL file for users, while in the ActiveBPEL Designer users have to manually create this WSDL file.
- To connect two Web services together, the outputs of the first web service have to be fed to the inputs of the second web service. In our WS-BioZard, this is handled automatically the by data mediation subsystem. In the ActiveBPEL Designer, users have to do it by themselves, such as define variables, specify the <assign> element to copy matched output of the first Web service to the input of the second Web service. In Figure 7, we can see five <assign> elements are used. In Figure 6 of WS-BioZard, the user will not be bothered by the <assign> elements; they are generated automatically. Although in the ActiveBPEL Designer, user can use wizard to do it, our automatic approach will be easier for biologists.
- WS-BioZard can deploy the created BPEL process to server automatically. However, in ActiveBPEL, there are two steps to deploy a BPEL process:
 - First, create a Process Deployment Descriptor (.pdd) file using wizard.
 - Second, export a Business Process Archive (.bpr) file that contains one or more BPEL processes to the execution server.

Usability:

- All the automatic aspects mentioned above will reduce the users' burden and therefore, increase the usability of our WS-BioZard.
- Comparing Figure 7 to Figure 6, our BPEL process is much simpler and focuses more on the business flow. Users only have to choose the Web services they want to connect together and WS-BioZard will take care of the details of how to connect them. In Figure 7, for the ActiveBPEL Designer users have to understand the details of BPEL specifications such as variables, <assign>/<copy>, XPath and why they need <receive>/<reply> at the ends of the process.

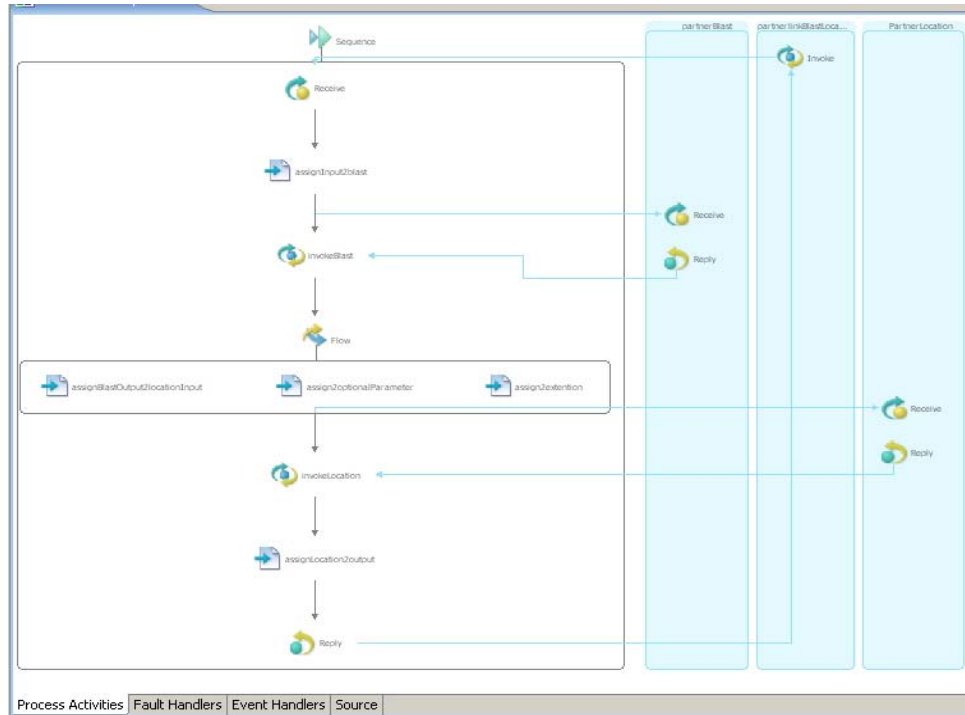


Figure 7. Workflow Made by ActiveBPEL

6.2. Sun's NetBeans BPEL Designer

In the NetBeans 6.0.1 release, the BPEL Designer is one component as is shown in Figure 8. Similarly to ActiveBPEL, it provides a visual editor to compose and edit BPEL processes. Our WS-BioZard is more automatic and easier to use compared to NetBeans BPEL Designer, although NetBeans BPEL Designer provides more automation compared to ActiveBPEL.

Automation:

- The NetBeans BPEL Designer also cannot automatically generate the WSDL file. Hence, our WS-BioZard which can automatically create the WSDL file provides more automation.
- NetBeans has a GUI tool for users to drag and drop and then generate the XPath used in the <assign> elements. It is helpful for users to manually finish the data mediation when users try to connect two Web services together. However, in WS-BioZard, data mediation is done automatically.
- Both NetBeans and WS-BioZard can automatically generate deployment descriptors and automatically deploy BPEL processes. In this regards, both NetBeans and WS-BioZard provide more automation than ActiveBPEL.

Usability:

- As mentioned above, although NetBeans has an excellent GUI to help users generate XPath and automatically deploy BPEL processes, users still have to know the details of the BPEL specifications. Thus, NetBeans is similar to ActiveBPEL in this respect and users have to link the input and output data manually, so WS-BioZard which handles data mediation automatically should be easier for biologists to use.

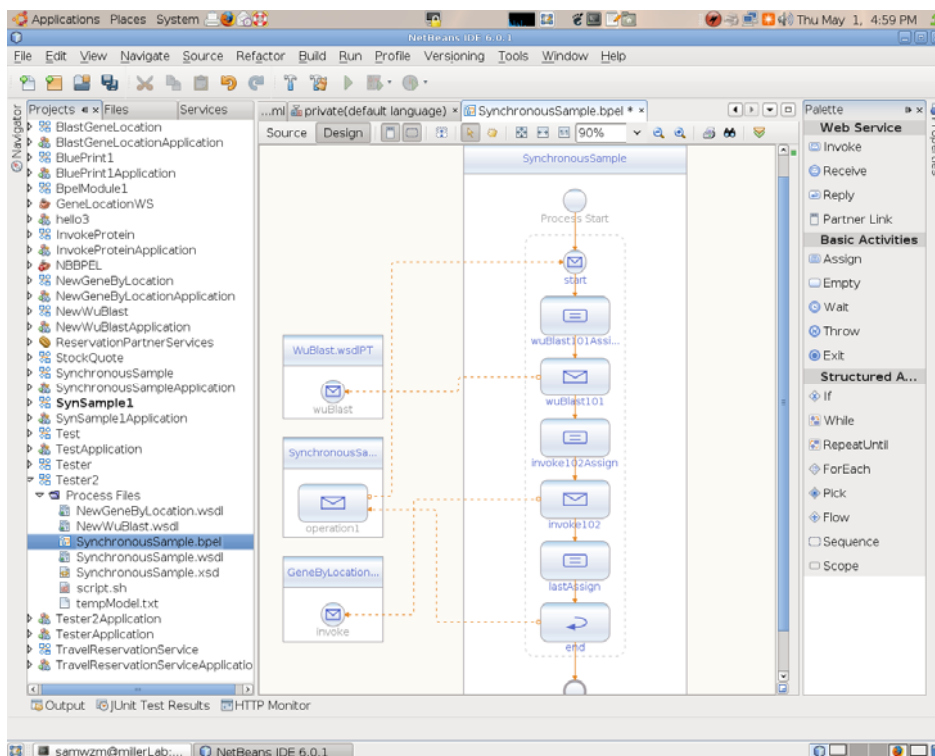


Figure 8. Workflow Made by NetBeans

6.3. Taverna's Designer

The Taverna workbench, created by the myGrid project, is a free software tool for designing and executing workflows. Taverna allows users to integrate many different software tools, including Web services. The Taverna Workbench provides a desktop authoring environment and execution engine for scientific workflows expressed in XScufl (XML Simple Conceptual Unified Flow language). Therefore, the Taverna Workbench will not create BPEL processes. We compare it here because the Taverna Workbench is a popular tool in biology and bioinformatics domain.

Automation:

- Since the Taverna Workbench is not a BPEL designer, most of the aspects of automation we compared above are not applicable to Taverna because there are no `<assign>`, `<receive>` or `<reply>` to be generated and no deployment at all.
- To help handle data mediation, Taverna provides “splitters” to parse the messages exchanged between two Web services. In Figure 9, the two green rectangles are the two Web services that we try to connect together. The purple

rectangles are all the splitters. Each splitter only can parse one layer down for one complex type element. That means the number of splitters to use depends on the number of complex type elements in the XML schema for the Web services' messages. Although the splitter is helpful, the biologists still have to have enough XML schema knowledge and manually finish the data mediation task. WS-BioZard can handle data mediation automatically, so it has higher automation and is easier to use.

Usability:

- Since BPEL is the main standard for specifying Web service processes, it is a disadvantage that Taverna does not use it. At some point, they may choose to replace XScufl with BPEL. The two main reasons for not doing this are that, as we have discussed, BPEL is complex and the changeover would require substantial re-coding.
- Since the workflow made by Taverna workbench is not a Web service, no deployment is required in Taverna, but at the same time it leads to less generality and accessibility. However, WS-BioZard creates the BPEL process which can be deployed as a Web service.

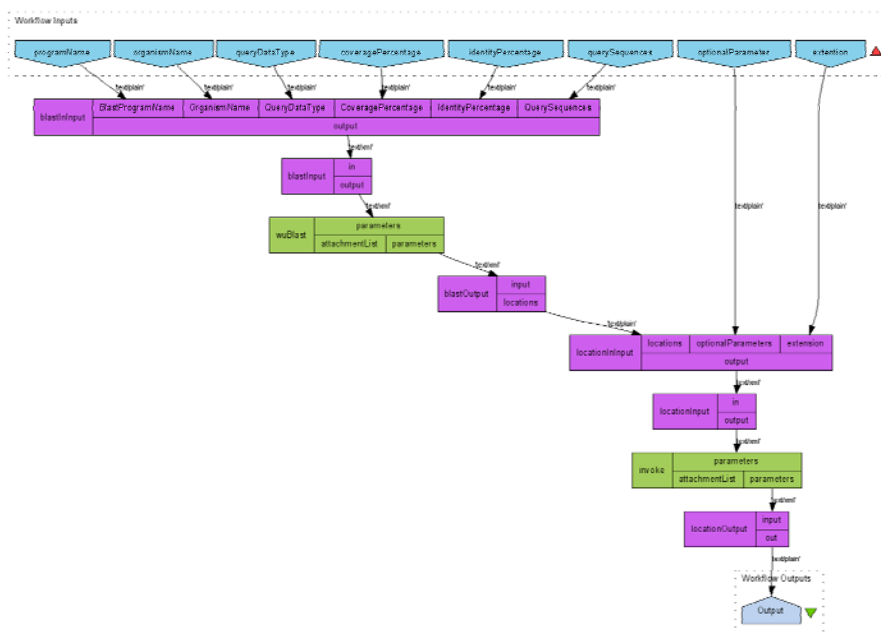


Figure 9. Workflow Made by Taverna

7. Conclusions and Future Work

With more and more bioinformatics Web services available now, the need for composing Web services to achieve complex tasks is increasing. Unfortunately, composing Web services is still too complex to be widely practiced by average biologists. Our framework addresses this problem and uses multiple technologies to lower the complexity of service composition. This framework is predicated on the view that design tools based on the concept of semi-automatic composition can effectively reduce the complexity of building BPEL processes, by utilizing automation where possible

and supplementing it with wizards to guide humans in designing the parts that currently do not yield well to automation. The framework is further predicated on the view that providing some level of semantic specification is essential. The new SASWDL standard along with supporting tools such as Radiant and Lumina, provide a low-cost in terms of time and effort, potentially high-gain approach to adding semantics to Web service descriptions, and thereby facilitating more precise discovery as well as opportunities to automate some aspects of composition.

WS-BioZard is a prototype implementation of this framework, which required the development of new techniques as well as their careful combination with existing Web service technology. An algorithm based on bottom-up semantic annotation of Web services to achieve data mediation automatically was developed. In any composition system, data mediation may be one of the biggest challenges and WS-BioZard provides a practical solution for this challenge. With a growing number of bioinformatics Web services emerging, manually finding and locating a suitable service is not a trivial task. With the aid of the semantic service discovery tool, the end user can query the available Web services published in a semantic UDDI registry through many criteria. WS-BioZard's innovation here was to reuse the discovery API provided with Lumina and develop a new GUI-based front-end that is particularly suitable for biologists. It makes it easy to sift through and visualize how related services can fit together in a composition.

Wizards that guide humans in performing computerized tasks are popular and effective. They are included in WS-BioZard in a parametric way in that their expertise is largely derived from a domain ontology. WS-BioZard includes a multi-faceted wizard serving as a biologist's assistant, helping him/her to design the BPEL process. The use of domain ontology not only provides expertise, but it does so in the language/terminology of the biologist using the tool. Use of the multi-faceted wizard allows a simplified BPEL editor to be presented to the user, and thus, lowers the complexity of composing BPEL processes. In this simplified BPEL editor, many BPEL elements such as "variable", "invoke", "assign", "partnerLink", "receive" and "reply" are hidden from the users, permitting them to focus more on business logic and not get overwhelmed by low-level details. WS-BioZard can automatically generate XPath expressions to aid in data mediation and condition expression. XPath is the default language used in BPEL for data access/manipulation and writing XPath manually is both difficult and error-prone.

There are many detailed steps required to make a BPEL process runnable. Leave any step out and you're stuck, so a tool suite for biologists should handle them all. Currently, all of the mainstream BPEL editors use a similar approach—defining the BPEL process WSDL before actually creating the BPEL process or even leaving it up to the designers to create the process WSDL themselves. Since it is not a trivial task for biologists to create a WSDL specification, it is important for WS-BioZard to automatically generate the process WSDL from the BPEL definition. To further simplify BPEL deployment, WS-BioZard can automatically deploy the BPEL process into Sun's Glassfish BPEL execution engine with the help of automatically generated deployment descriptors. Finally, after a BPEL process has been deployed into an execution engine, an intuitive GUI client can be popped-up to invoke the deployed BPEL process as a standard Web service.

In the future, the service discovery tool should be more tightly integrated with the service composition editor. That is, the

user should be able to drag-and-drop the service selected in the service discovery tool into the service composition editor. The icons of the discovered services should remain in the composition editor, in a sense, turning the composition problem into a puzzle solving problem (how to make the color coded services icons fit together). To make WS-BioZard more generally useful, its customization features (particularly for the discovery tool) should be improved and tested on multiple ontologies. Finally, further wizard capabilities should be added to assist in the creation of more advanced process flow constructs such as loops and parallel execution. Once these enhancements have been made, WS-BioZard should be quantitatively evaluated to determine (i) the degree to which it lowers the learning curve for composition, (ii) how much it increases a designer's productivity in creating BPEL processes and (iii) how it helps biologists by saving them time and effort.

Acknowledgments

Z.W. was supported by NIH R01 AI058515 to J.C.K. We would like to thank all the ApiDB project team members for their help in this study. J.A, J.C.K, D.B and C.A has been funded in whole or in part with Federal funds from the National Institute of Allergy and Infectious Diseases, National Institutes of Health, Department of Health and Human Services, under Contract No. HHSN266200400037C.

References

- [1] F. Casati, S. Ilnicki, and L. Jin. Adaptive and dynamic service composition in EFlow. In Proceedings of 12th CAiSE, June 2000.
- [2] H. Schuster, D. Georgakopoulos, A. Cichocki, and D. Baker. Modeling and composing service-based and reference process-based multi-enterprise processes. In Proceeding of 12th International Conference on Advanced Information Systems Engineering (CAiSE), Stockholm, Sweden, June 2000. Springer Verlag.
- [3] F. Casati, M. Sayal, and M.-C. Shan. Developing e-services for composing eservices. In Proceedings of 13th International Conference on Advanced Information Systems Engineering(CAiSE), Interlaken, Switzerland, June 2001. Springer Verlag.
- [4] R. Hull, M. Benedikt, V. Christophides, and J. Su. E-service: A look behind the curtain. In Proceeding of the 22th PODS, San Diego, USA, June 2003.
- [5] J. Rao and X. Su. "A Survey of Automated Web Service Composition Methods". In Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, SWSWPC 2004, San Diego, California, USA, July 6th, 2004.
- [6] C. Peltz. "Web Services Orchestration and Choreography", *Web Services Journal*, Volume 03--07, July 2003, pp. 30-35
- [7] Ranjit Mulye. "METEOR-S Process Design and Development Tool" Master These
- [8] W. M. P. van der Aalst and A. H. M. ter Hofstede. YAWL: yet another workflow language. *Information Systems*. Volume 30, Issue 4, pp. 245-275, 2005.

- [9] E. Pignotti, P. Edwards, A. Preece, G. Polhill and N. Gotts. Semantic Workflow Management for E-Social Science. Proceedings of the Third International Conference on eSocial Science, October 2007.
- [10] Workflow Patterns Standards Evaluation, www.workflowpatterns.com/evaluations/standard/index.php
- [11] C.A. Goble, R. Stevens, G. Ng, S. Bechhofer, N.W. Paton, P.G. Baker, M. Peim, and A. Brass. Transparent Access to Multiple Bioinformatics Information Sources. *IBM Systems Journal*. Special issue on deep computing for the life sciences, 40(2):532 -- 552, 2001.
- [12] S. McIlraith, and T. Son. Adapting Golog for Composition of Semantic Web Services. In Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning (KR2002), pp. 482—493, 2002.
- [13] Dan Wu, Bijan Parsia , Evren Sirin , James Hendler , and Dana Nau, “Automating DAML-S Web Services Composition Using SHOP2”, ISWC 2003, LNCS 2870, pp. 195-210, 2003.
- [14] B. Srivastava, J. Koehler, “Web service composition: Current solutions and open problems”, ICAPS 2003 Workshop on Planning for Web Services, 28—35, 2003
- [15] L. T. Santos, P.A. Roberto, M.A. Gonçalves, H. F. Laender: Design, Implementation, and Evaluation of a Wizard Tool for Setting Up Component-Based Digital Libraries. *Research and Advanced Technology for Digital Libraries*.
- [16] C. Petrie. The World Wide Wizard of Open Source Services. 2007, IEEE International Conference on Web Services (ICWS' 2007)
- [17] M. Nagarajan, K. Verma, A.P. Sheth and J.A. Miller, "Ontology Driven Data Mediation in Web Services," *International Journal of Web Services Research*, Vol. 4, No. 4. 2007.
- [18] K. Verma, K. Sivashanmugam, A.P. Sheth, A. Patil, S. Oundhakar and J.A. Miller, "METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services," *Journal of Information Technology and Management*, Special Issue on Universal Global Integration, Vol. 6, No. 1. 2005.
- [19] E. Sirin, B. Parsia, and J. Hendler. Filtering and Selecting Semantic Web Services with Interactive Composition Techniques. *IEEE Intelligent Systems*, 19:42–49, 2004.
- [20] M. Xu, J. Chen, Y. Peng, X. Mei and Ch. Liu. A Dynamic Semantic Association-Based Web Service Composition Method. *Web Intelligence*, Issue18-22, 2006.
- [21] J. Kim, M. Spraragen, and Y. Gil. An Intelligent Assistant for Interactive Workflow Composition. In IUI '04: Proceedings of the 9th international conference on Intelligent user interface, pp. 125–131, New York, NY, USA, 2004. ACM Press.
- [22] F. Hakimpour, D. Sell, L. Cabral, J. Domingue, and E. Motta. Semantic Web Service Composition in IRS-III: The Structured Approach. In 7th IEEE International Conference on E-Commerce Technology. pp. 484–487, 2005.
- [23] R.D. Stevens, A.J. Robinson, and C.A. Goble. my Grid: personalized bioinformatics on the information grid. *Bioinformatics*, 19(90001):302i–304, 2003.
- [24] M.D Wilkinson and M. Links. BioMOBY: an open source biological web services proposal. *Brief. Bioinform.*, 3, 331–341, 2002.
- [25] Y.C. Song, E. Kawas, B.M. Good, M.D. Wilkinson and S.J. Tebbutt. DataBiNS: a BioMoby-based data-mining workflow for biological pathways and non-synonymous SNPs. *Brief. Bioinform.*, 23, 780-782, 2007.
- [26] Kawas E, et al. BioMoby extensions to the Taverna workflow management and enactment software. *BMC Bioinformatics*, ((2006)) 7, : 523.
- [27] Halevy, A. 2005. Why Your Data Won't Mix: Semantic Heterogeneity, *ACM Queue*

- [28] Litwin, W. and Abdellatif, A., 1986, Multi-database Interoperability, *IEEE Computer*, 19(12). 10-18.
- [29] Sheth, A., 1998, Changing Focus on Interoperability in Information Systems: From System, Syntax, Structure to Semantics, *Interoperating Geographic Information Systems*. 5-30.
- [30] Kunal Verma, Kaarthik Sivashanmugam, Amit P. Sheth, Abhijit Patil, Swapna Oundhakar and John A. Miller, "METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services," *Journal of Information Technology and Management (ITM)*, Special Issue on Universal Global Integration, Vol. 6, No. 1 (January 2005) pp. 17-39.
- [31] Michi Henning, The rise and fall of CORBA, *Queue*, v.4 n.5, June 2006 [doi>10.1145/1142031.1142044]
- [32] R. Chinnici et al., "Web Services Description Language (WSDL) Version 1.2," *World Wide Web Consortium*, 2002, www.w3.org/TR/wsdl12/.
- [33] D. Box et al., "Simple Object Access Protocol (SOAP) 1.1," *W3C Note 08*, *World Wide Web Consortium*, May 2000, www.w3.org/TR/SOAP/.
- [34] Muehlen, M., Nickerson, J.V. and Swenson, K.D. (2005), "Developing web services choreography standards – the case of REST vs. SOAP", *Decision Support Systems*, Vol. 40, pp. 9-29.
- [35] F. Curbera et al., "Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI," *IEEE Internet Computing*, vol. 6, no. 2, 2002, pp. 86—93.
- [36] G. Kandaswamy et al., "Building Web Services for Scientific Applications," *IBM J. Research and Development*, vol. 50, no. 2/3, 2006, pp. 249–260.
- [37] Ping Jiang , Quentin Mair , Julian Newman, Using UML to Design Distributed Collaborative Workflows: from UML to XPDL, *Proceedings of the Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, p.71, June 09-11, 2003
- [38] F. Leymann, *Web Services Flow Language (WSFL) 1.0*. IBM Software Group, May 2001.
- [39] S. Thatte *XLANG: Web Services For Business Process Design*, Microsoft Corporation (2001)
- [40] S. Narayanan, S.A. McIlraith, Simulation, verification and automated composition of web services, in: *Proceedings of the Eleventh International World Wide Web Conference (WWW-11)*, Honolulu, Hawaii, USA, May 2002.
- [41] Aurrecoechea, C., Heiges, M., Wang, H., Wang, Z., Fischer, S., Rhodes, P., et al. (2007) *ApiDB: integrated resources for the apicomplexan bioinformatics resource center*. *Nucleic Acids Res* 35: D427–D430.
- [42] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, D.F. Ferguson (Editors). *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall, 2005.
- [43] A. Barros, M. Dumas, and P. Oaks. A critical overview of the web services choreography description language(ws-cdl). *Business Process Trends*, 2005.
- [44] A. Arkin. *Business Process Modeling Language (BPML)*, Version 1.0. BPML.org, 2002.
- [45] *OMG UML 2.0 Specifications*. <http://www.omg.org/uml/>
- [46] E. Newcomer and G. Lomow. *Understanding SOA with Web Services*. International Thomson Computer Press, 2004.
- [47] James Clark and Steve DeRose (eds.). *XML path language (XPath) version 1.0*. W3C Recommendation.
- [48] Zixin Wu, Karthik Gomadam, Ajith Ranabahu, Amit P. Sheth and John A. Miller, "Automatic Composition of Semantic Web Services using Process and Data Mediation," *Proceedings of the 9-th International Conference on Enterprise Information Systems (ICEIS'07)*, Funchal, Portugal (June 2007) pp. 453-461.

Using these new high-level Eclipse frameworks, a developer can create a sophisticated GUI with a fraction of the effort that would have been required in the past. This also allows the developer to focus on the application logic and on making software easy to maintain and extend. This infrastructure can provide the following benefits:

- Focus more on high level concerns—since low level graphical part can be automatically generated by GMF.
- Faster development—utilizing the visual design environment and source code generation.
- Fewer bugs—by reducing manual coding compared with pure Eclipse GEF development.
- Improved maintenance—facilitated by well organized package structure.
- Easy extension both on function part and behavior part—for function part, only thing to do is modify the GMF model definition and regenerate source code; for behavior part, it is also easy to achieve thanks to the design patterns used in EMF.