Cluster Computing 8, 279–291, 2005

A Dynamic Workflow Approach for the Integration of Bioinformatics Services

MALIKA MAHOUI
School of Informatics, IUPUI, Walker Plaza, 719 Indiana Avenue, Indianapolis, IN 4620, USA

LINGMA LU
Department of Computer and *Information Science, IUPUI*

NING GAO and NIANHUA LI
Department of Electrical and Computer Engineering, IUPUI

JESSICA CHEN
School of Informatics, IUPUI, Walker Plaza, 719 Indiana Avenue, Indianapolis, IN 4620, USA

OMRAN BUKHRES
Department of Computer and Information Science, IUPUI

ZINA BEN MILED
Department of Electrical and Computer Engineering, IUPUI

Abstract. Modern biological and chemical studies rely on life science databases as well as sophisticated software tools (e.g., homology search tools, modeling and visualization tools). These tools often have to be combined and integrated in order to support a given study. SIBIOS (System for the Integration of Bioinformatics Services) serves this purpose. The services are both life science database search services and software tools. The task engine is the core component of SIBIOS. It supports the execution of dynamic workflows that incorporate multiple bioinformatics services. The architecture of SIBIOS, the approaches to addressing the heterogeneity as well as interoperability of bioinformatics services, including data integration are presented in this paper.

Keywords: bioinformatics, service integration, web-based applications, dynamic workflow

1. Introduction

The recent availability of high throughput technologies has resulted in a large amount of data. Knowledge is extracted from this data by using various multi-step computational methods. Several tools are available to support knowledge extraction. Example tools include database search tools, homology search tools (e.g. PSI-BLAST [2]) and clustering tools. In the remainder of this paper, the name services will be used to refer to these software tools. Marshalling these services in order to construct a complete biological study remains a challenge because of the lack of interoperability among the services. Consider the example given in [29] describing the steps involved in determining the protein family of a protein that corresponds to a given DNA sequence. This task requires a series of elementary steps (figure 1). First, the sequence that corresponds to the unknown DNA fragment is processed through a translator such as Transeq [4] in order to extract the six possible reading frames. The second step consists of identifying the correct reading frame by searching a composite database such as OWL database [28], and using each of the six possible reading frames as input until a hit is found. Once the appropriate peptide is identified, the corresponding sequence is used to search a protein database such as PIR [30] in order to retrieve the entire protein sequence. During the fourth step, the protein family is identified by using a homology search service such as PSI-BLAST. The fifth step consists of obtaining the function of the sequence by searching several databases such as Prosite [31], Profiles [32], Pfam [33], eMOTIF [15] and BLOCKS [10] in order to locate information about the target protein. These databases can be searched concurrently. Currently, the process underlying this example is performed manually by scientists. In addition, the output of one service has to be manually manipulated in order to be used as input for another service. Finally, data from two or more services may have to be semantically integrated by resolving heterogeneities in the output of the various services. These factors make the execution of tasks involving more than one service time consuming and error prone.

One of the major obstacles in the integration of services is their heterogeneity. When services are limited to database searches, several successful approaches [4,13,16,37] to the integration of life science databases have been proposed. Addressing the more general case of service integration

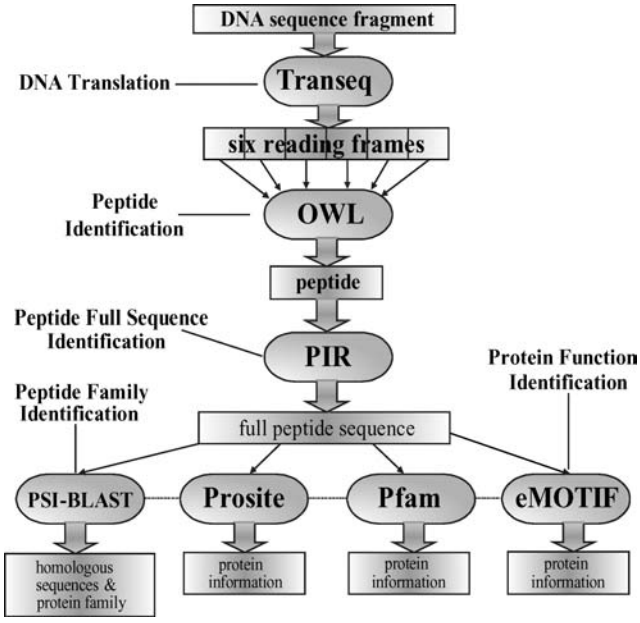


Figure 1. Searching for protein family information for an unknown DNA fragment.

is more challenging. Several service integration approaches [14,16,17,35,38,45,46] have been proposed. There are three main aspects that distinguish these approaches: service discovery, workflow capabilities and heterogeneity resolution.

Service discovery allows the user to select the service that can perform a given task. For example, a homology search can be performed by using Blast [2] or Fasta [34] among others. Furthermore, there may be several different versions of a given tool. For instance, Blastn and Blastp are two versions of Blast that can perform homology searches for nucleotide sequence and for protein sequence, respectively. Service discovery capabilities vary from a simple predefined list from which the user can select the desired service [35,38], to more enhanced capabilities which discover the appropriate service based on properties specified by the user [26,45]. Examples of descriptive properties include the name of the task the service is performing and the type of the input it accepts. The more intelligence is built in the service discovery the better because of the large number of available services and the lack of familiarity that the scientists may have with these services.

The second feature that characterizes service integration systems is the level of workflow capabilities or adaptability supported by the system. An integration system with workflow capabilities allows the specification of the services that compose a biological task and their sequence of execution a priori. Systems such as SRS [13] are not workflow-based systems. SRS allows the specification of one service at a time. Based on the results generated by a service, the following service can be selected in SRS. A workflow can either be static or dynamic. Static workflows are fully automated, whereas dynamic workflows are partially automated. In the latter case, user can intervene during the workflow execution and the workflow is highly adaptable. Almost, all service integration systems that include workflow capabilities such as [21, 42, 44] only support static workflow execution.

The third feature of service integration systems is whether semantic and syntactic data heterogeneities are addressed or not. Resolving data heterogeneities among the input and output data of the services increases the flexibility of the service integration system. Furthermore, in the life science domain and because of the extreme variances in the data representations and the semantic definitions of concepts adopted by the various services, ignoring this issue will render any workflow management system impractical.

SIBIOS, a System for the Integration of BIO-informatics Services, is proposed in this paper. It integrates both web-based and standalone services. Examples of services integrated by SIBIOS include database search services (e.g., Swiss-Prot database search [39]) as well as other software tools (e.g., BLAST). SIBIOS is based on client-server architecture. On the server side, the task engine supports the invocation of the distributed services as well as mitigates any interoperability issues among the services.

Section 2 presents the research issues addressed in this paper. The architecture of SIBIOS is presented in Section 3. The dynamic workflow model supported by SIBIOS is discussed in Section 4 and implementation details of the system are provided in Section 5. Related work is discussed in Section 6. Section 7 includes a summary of the features of SIBIOS and directions for future work.

2. Research challenges

SIBIOS is an integration system for bioinformatics services and as such its design raises several unique research issues that are related to the features of bioinformatics services and the conditions of their deployment in a workflow. This section explains these issues and highlights the solutions adopted by SIBIOS.

2.1. Service discovery

SIBIOS is an integration system for bioinformatics services and as such its design raises several unique research issues that are related to the features of bioinformatics services and the conditions of their deployment in a workflow. This section explains these issues and highlights the solutions adopted by SIBIOS.

Services are the basic components used to build a workflow. The process of finding the right service that can accomplish a given task in a workflow is called *service discovery*. This process is particularly challenging, as the number of available databases and bioinformatics tools is very large. Classification, which consists of arranging the services into related groups, becomes a necessary step in order to facilitate the process of service discovery. The use of service functionality for service classification has been widely adopted by existing bioinformatics service integration systems (e.g., EMBOSS

[35] and VIBE [21]). Other systems support additional service discovery options; and taxonomies are used within each option to hierarchically structure services and therefore facilitate service discovery [45,47]. For example, in Biomoby [45] searching for services can be based on the input type or the output type of a service. The availability of different classifications, which are based on the descriptive features of the services, clearly provides more alternatives when searching for a service. Additional flexibility can be gained by allowing the combination of several of the service discovery options. This will allow a more refined service discovery process, which is particularly important when the number of services is large, as in the case of the bioinformatics field.

SIBIOS exploits the properties used to describe services for service discovery, thus supporting very refined and flexible service discovery. For example, the description of a service such as Blastn includes the type of input parameters that it accepts (i.e. nucleotide sequence) and the task it performs (i.e. homology search). These two properties, namely *has_input* and *performs_task*, are provided, and can be used by the user for service discovery. The domain of each property is also hierarchically structured to further refine the service discovery process. For example, *Nucleotide sequence* is hierarchically classified under *sequence*, which in turns is classified under *physical structure*. Therefore, when a service with *has_input sequence* is desired by the scientist, all the services whose input falls under the classification *sequence* such as *nucleotide sequence* will also be retrieved. Each service property is used independently from the others to classify services. However, they can be used individually or combined to specify the desired service.

The classification of service in order to facilitate service discovery can be either static or dynamic. Static service classification is used in Biomoby [45] while dynamic discovery classification has been introduced by Mygrid [47] and in Keppler [1]. In a static classification, services that share the same property are grouped and stored together. This information is redundant because it is derived from the information used to describe the services in a service integration system. Additional redundancy can also be due to the fact that a service may belong to more than one classification. For example, the service Blastn, which was described earlier, will belong to at least two classifications, one associated with the property *has-input* and the other associated with the property *perform_task*. In addition to redundancy, another drawback of static classification is that each time a new service is added various classifications may need to be reorganized.

In a dynamic classification, there is no redundancy since during the service discovery process, the description of services is directly used to infer the name of the services that match a given property. SIBIOS uses dynamic service classification to support service discovery. The description of the input, output and capability of each service in SIBIOS is included in a service schema file. Each service is associated with a schema file. The descriptions used in the schema files are based on concepts extracted from ontology. The ontology in SIBIOS does not include any service specific information and is therefore relatively stable. The service schema file in conjunction with the ontology is used to infer service classification dynamically. This approach compared to the classification of service is scalable, flexible and easily maintainable. For example, adding a new service descriptive property will automatically generate a new service classification. Furthermore, the addition of new services does not necessitate any rebuilding of the service classification because this classification is generated dynamically during service discovery.

2.2. Workflow capabilities

The complexity of biological studies is often based on a process that consists of multiple services [5,21,38]. In traditional workflows, the objective is to automate a process according to a set of procedural rules [43]. Scientific workflows are different from traditional workflows because they are exploratory-based. The experimental nature of bioinformatics services requires an iterative workflow construction procedure. The workflow specification may be iteratively adjusted by changing the services in the workflow and by manipulating the output of one service before feeding it into the input of another service. The importance of adaptive workflows that can rapidly respond to context-sensitive information changes or external events in scientific domains is discussed in details in [24]. Once the workflow becomes stable, it can become part of the laboratory procedure and used in a way similar to other experimental protocols. The distinctions between the experimental phase and the production phase result in two different modes of execution for the workflow. Both of these modes are supported in SIBIOS and the support for this adaptability is dictated by a survey of a wide range of real life science applications.

Most of the previously proposed workflow systems are also subject specific such as a workflow for protein-protein interaction [24]. SIBIOS provides a general framework that can be easily adapted to any given subject, such as the studies of system biology, pathway, protein structure, proteomics, etc. Furthermore, traditionally, workflow systems ignore the issues related to data integration when enacting multiple services. SIBIOS addresses this issue and thus, provides a comprehensive data and service integration solution.

2.2.1. Dynamic workflow execution

In order to enable the exploratory workflow construction, it is important to allow the user to interact with the system during the execution of the workflow. For example, it can eliminate the unnecessary execution of services subsequent to a service that is deemed not to deliver the required output. In the example of figure 2, the user has no a priori knowledge of the sequence(s) to be selected as the input for the Pfam service. The workflow can be executed without interruption until it reaches the Pfam service. User intervention will then allow the selection of the appropriate sequences from the result set produced by the Swiss-Prot service that will be used as the input to Pfam. Additional actions that the user may undertake

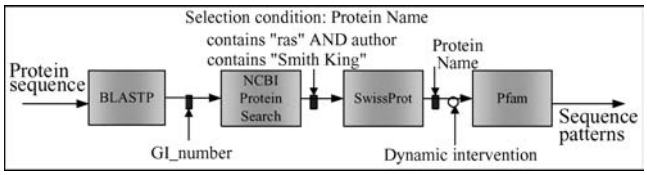


Figure 2. Example of a dynamic workflow.

during an intervention include adding and dropping services as well as modifying the data or the flow control among the services.

2.2.2. Data filtering capabilities

Data filtering allows the selection of a subset of the output data from one service to be used as the input of a subsequent service. It is particularly important for bioinformatics for two reasons: elimination of unneeded output fields in an output record and elimination of irrelevant records. In traditional applications such as a business application, the output of a service is generally limited to a small number of output types (e.g., a currency exchange service executed on two currencies delivers one type of output - the currency exchange rate). Whereas bioinformatics services produce a large number of output types, including cross-references to information of other services. Consider for example a workflow where the BLASTN service follows a Swiss-Prot search service. The results output from the latter service include *Swiss-Prot entry name*, *Swiss-Port accession number*, *protein name* and *protein sequence*. However, only the protein sequence is used as input to BLASTN. Unnecessary information can be eliminated by filtering the output data from Swiss-Prot, which is referred to by projection in relational database area.

The elimination of irrelevant records is also important in the integration of bioinformatics services, which tend to be based on heuristics that may return false positives. Data filtering conditions can be used to automatically select a subset of the records, similar to selection in relational databases area. Consider for example the execution of an NCBI Entrez search [32] on a set of *GI numbers* produced by a previous service (figure 2). Among the records returned, only those related to the "*ras*" species and for which the author of the publication is "*Smith Kin*g" are of interest to the scientist. A selection condition that specifies that *protein name* should include "*ras*" and author should include "*Smith King*" will extract these records.

2.2.3. Support for logical expressions

Most database search services (e.g., PIR, Swissprot, and NCBI entrez) support queries that include logical operators. Moreover, in a typical workflow scenario, the input query to a database search service is automatically determined by using the output of one or more preceding services. Thus, workflow specification should support logical expressions. For example, consider the execution of a Swissprot search service on the output produced by a GenBank [33] search service. This output includes the *NCBI accession number* and *organism name*, which can be combined by a logical expression (e.g., "NCBI accession number OR organism name") and fed to the Swissprot search service. Specifically, if one of the records generated by GenBank has an accession number = NC_001136 and an organism name = Saccharomyces cerevisiae, the query that will be submitted to the Swissprot search service will be based on the following URL which includes a Boolean expression: http://us.expasy.org/cgi-bin/ sprot-searchful?makeWild = & SEARCH = NC_001136%20or%20Saccharomyces%20or%20cerevisiae

2.2.4. Iterative execution of services

In a workflow model, an iterative (i.e., recursive) service may be executed until a termination condition is satisfied. For example, consider the execution of the FASTA service [34] on a protein sequence. A homologous sequence returned by the service is considered for further analysis only if its corresponding expectation value is less than a certain value "*v*". The FASTA service allows the user to set a threshold value "*v*" for the expectation value. However, if the service returns an empty result, FASTA needs to be iteratively invoked on the same input sequence until the expected value condition is met. For this iterative procedure to work properly, each time when the service is invoked, certain environment parameters such as the gap penalty or the scoring matrix need to be changed (e.g., the gap penalty can be halved at each iteration until the threshold value condition is met).

2.2.5. Join and Union of service results

In a workflow, the result of more than one service can be combined either by a *join* or a *union* operation and used as the input of next service. In figure 3(a), two homology search services (i.e., FASTA and BLASTP) produce different results starting from the same input protein sequence. Performing the homology search by using different tools is useful because different homology search services may have different levels of sensitivity (i.e., ability to detect distant homologies). More

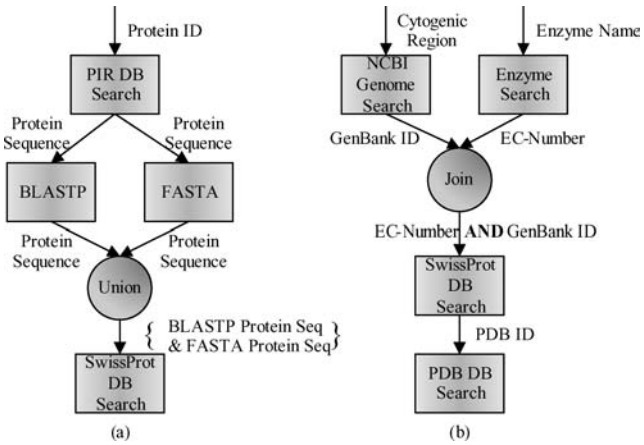


Figure 3. Examples of workflows with *union* (a) and *join* (b) operations.

importantly, the homology search services may operate on different databases and it is not known a priori which database may include the important homologues [36]. Therefore, multiple homology search services can be run concurrently and their results can then be combined by using a *union*, which will result in a list of unique sequences generated by all the services.

Figure 3(b) shows a different way in which the results from two services can be combined using the *join operator*. The NCBI genome database search service [20] provides the record ID (GenBank ID) of genes located within human chromosome region 4q21-4q23, whereas Enzyme database search service [6] provides the enzyme class ID (EC-number) of alcohol dehydrogenases. Therefore, the Cartesian product of the above two result sets can identify all the alcohol dehydrogenases located within the human chromosome region 4q21-4q23 in Swiss-Prot database search service. Specifically, if one of the records generated by the Enzyme database search has an EC-number = 1.1.1.1 and one of the records generated by NCBI genome database search has a GenBank ID = M12963, then the query that is submitted to the Swiss-Prot search service will be based on the following URL: http://us.expasy.org/cgi-bin/sprot-search-ful?SEARCH = M 12963 + AND + 1.1.1.1 &S = on.

2.3. Data integration

It is necessary for a workflow system to resolve interpretabilities among heterogeneous services regardless of service types and input/output format. In order to support data exchange, data has to first be extracted from the service and then mapped to a uniform semantic data model.

2.3.1. Data extraction

In business domain, web service description languages such as WSDL [7,12] are widely employed to facilitate data extraction. However, no such acceptance has been observed in the biomedical research field. Although it counts web-based applications in the hundreds, the large majorities are not defined as web services. Many integration systems construct service specific wrappers for data extraction. This approach has several disadvantages: (1) domain expertise is required for wrapper construction and maintenance. (2) a large number of wrappers has to be written to accommodate the large number of available bioinformatics services (3) wrappers are very sensitive to small changes that occur in the web pages which require wrappers be updated regularly.

SIBIOS adopts an approach that attempts to reduce these challenges. This approach was successfully used in BACIIS [11], a system that integrates life science web databases. The main idea is to separate individual service description from the wrapper engine. The service description is stored in a file called the service schema. This file includes domain specific knowledge described by using ontology and a set of rules that describe how the data can be extracted from the services. The wrapper engine reads these service schema files and generates data source specific wrappers dynamically. More information about service schema locates in the Section 4.4.

2.3.2. Semantic integration

Bioinformatics services are highly heterogeneous in data format, terminology, concept semantics, content structure, etc. [39]. Ontologies are widely used for the semantic integration of heterogeneous services. For example, MyGrid [47] describes bioinformatics web services in an ontology by using the DAML + OIL language. MyGrid is a framework for service integration. SIBIOS uses a similar approach where the ontology serves as a common data model for the services.

The problem of semantic heterogeneities is resolved at two levels in SIBIOS: during workflow construction and during workflow execution. The workflow construction ensures the input of each service in the workflow matches the output of its preceding services. It is achieved by describing the input/output data type of each service through ontology. Two services can be connected if their input/output data correspond to the same ontology term. Semantic heterogeneity is also resolved during workflow execution. The schema file, which is used by the wrapper engine, describes the corresponding service using the ontology concepts. Data is extracted from a given service and mapped to this common data model before it is processed and passed to the next service.

3. SIBIOS architecture

SIBIOS uses client-server architecture as shown in figure 4. On the client side resides two modules: *Workflow Builder* and *Result Manager*. The rest of the modules are located on the server side.

The user accesses the system through a user interface, which accepts the user query, displays the workflow, the results, as well as the status of the services during execution. The workflow builder communicates with the service discovery component to assist the user for service discovery. It also interacts with the schema wrapper (i.e. handler) engine to obtain detailed description of new services added to the workflow. Other functions of the workflow builder include the specification of how services are connected and what data processing is to be performed on the output of the services. Workflow information is displayed in a diagram on the client side, and converted by the workflow manager to an internal representation called workflow object on the server side. The workflows are executed by the task engine, and processing status is monitored by the process manager module. Remote services are specified by service schema wrapper. The results of each service are available to the user as soon as the service finishes execution. Users can manipulate the data set (e.g. filter) fed to the next service using the result manager.

Users may update a workflow during its execution. In this case, the task engine relinquishes the control back to the workflow builder. The execution will be resumed after the workflow

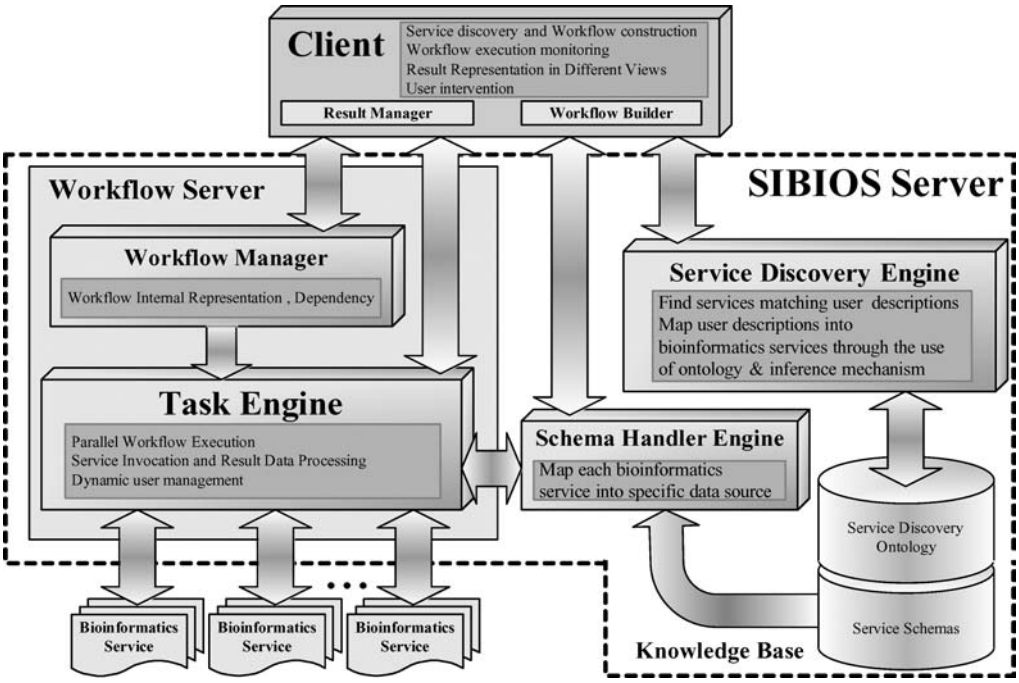


Figure 4. SIBIOS client-server architecture.

modifications are reflected on both the server and the client side. The internal representations of workflows are stored in a workflow repository. This feature allows the user to execute the same workflow multiple times or to use previously constructed objects to generate new workflow objects. The implementation details of the SIBIOS workflow component and the task engine are described in Section 5.

4. SIBIOS Dynamic Workflow Model (SDWM)

The standard components of a workflow process described by the WfMC's WPDL [30] are tasks (i.e., services) and transitions between tasks. In SDWM the specification of a service s is based on four parameters: input attributes $I(s)$, output attributes $O(s)$, environment parameters $E(s)$ and logical operators $L(s)$. The environment parameters are optional (e.g., the scoring matrix to be used by a BLAST service). The set of logical operators that can be used to form a logical input expression when executing s is denoted by $L(s)$. Formally a service is defined as follows:

Definition 1. a service s is the quadruplet *(I(s), O(s), E(s), L(s))*

If $L(s)$ is empty then only one element from $I(s)$ can be used as input during the invocation of s. In addition, we assume without loss of generality that the input data or the output data can be organized into records with multiple attributes following the relational database model where an attribute represents a data field in the input or the output.

Definition 2. The transition between two services $s1$ and $s2$ in S is defined by the function f such that $f(s1) = s2$ if $O(s1)$ and $I(s2)$ have at least one attribute in common.

The main features that support SDWM fall into three classes: filtering operators, connector operators, and service execution modes.

4.1. Filtering operators

Filtering operators, including selection, projection operators, or their combinations, are used for filtering service output in SIBIOS.

Definition 3. Let s be a service and o_i be an attribute in $O(s)$. A selection condition applied on o_i is a logical expression of the form "$o_i op\ v$", where op is an operator in $\{<, >, \leq, \geq, =, contains, not\text{-}contain\}$ and v is a value in the domain of o_i. For example, "protein-name *contains* '*ras*' " is a selection condition.

Definition 3 shows a primitive selection condition. More complex selection conditions can be obtained by combining multiple primitive selection conditions using Boolean operators. The selection operation applies the selection condition to the output of s and generates only the records that satisfy this condition.

The other filtering operation is projection, which will restrict the output of a service s to a subset of the attributes in $O(s)$:

Definition 4. $\text{Project}_u(s) = s1$ where $u \subset O(s)$ and $O(s_1) = u$.

A project operation selects data types that are only of interest to the user. Among the output types $O(s)$, only the set u is selected as the input types of a subsequent service.

4.2. Connector operators

Connector operator specifies the input of the service *s* through a logical expression based on the exported output of previous services. The exported output refers to the output after all filtering operations are performed and the purpose of the logical expression is to define the connection between two or more services.

There are three types of connectors: *primitive* connector, *union* connector and *join* connector. The *primitive* connector, which is depicted by the arrow between BLASTP and NCBI Protein Search in figure 2, is used when a service *s* has only one antecedent service. The *join* and *union* connectors, as illustrated in figure 3(a) and (b) respectively, are used when a service has more than one antecedent service.

4.3. Service execution modes

Services can be executed in three different execution modes: regular, dynamic and iterative. In the regular execution mode the service is to be executed once and its results are to be fed to the next service automatically without any interruption from the user. When the dynamic execution mode is set prior to a given service, workflow execution will pause before the execution of this service, thus allowing for user intervention. Execution will resume after the intervention.

When the iterative execution mode is associated with a given service, it will be executed more than once on the same input until a certain condition is satisfied. An example of an iterative execution mode is given in Section 2.2.4. In some cases, as in the example, it is necessary to change the environment parameters from one iteration to the next. This change can be done either automatically or manually. In a manual approach, the user needs to intervene each time the condition is not satisfied whereas the automatic approach relies on the specification of the changes that need to occur for each environment parameter. In the current model, only the first approach is used. Future work will investigate the second approach.

4.4. Service schema

Each service supported by SIBIOS has an entry in the ontology and an associated service schema file. The entry in the ontology is a high level description of the service that is used only during service discovery process. A more detailed description is included in the service schema, which is used by the workflow builder during workflow building and by the task engine during workflow execution. Workflow builder uses service schema to extract service input types and default values and the service output types. The former information is used to help the user set the service execution parameters (e.g. gap penalty for blastn is set to -14), while the latter information is used to specify the data filtering conditions (e.g. EC number contains "1.14.17.1"). The task engine uses the service schema during the execution of the remote service and during the processing of the output data.

The service schema is expressed in XML format. Service parameters are described using ontology concepts as XML tags, whereas tag values are service specific information about how the service is invoked and how the desired data can be extracted from the data record pages. Using the ontology concepts to describe the service helps in addressing syntactic and semantic variances among the services and thus, facilitates service integration.

Figure 5 shows part of the FASTA service schema. The information inside the SERVICE_APPLICATION section includes general information such as the service name (i.e. <service_name>), and the extraction rules that are used to retrieve the result pages (i.e. <path_page>). The URL_STRING section includes ontology concepts that are used to describe the input data (e.g., *database type*), including the default values for optional parameters; and the basic URL used to invoke the (http://www.ebi.ac.uk/cgi-bin/fasta/submit). The next section contains ontology concepts that describe the output data (e.g., *alignment_data*) and their corresponding extraction rules. Ontology terms such as *data_store* are mapped onto service specific terms such as *database*. Figure 5 also includes hierarchically structured ontology terms such as *summary_output* and *alignment_details*. The last section (i.e. INTERFACE_PARAMETER>) includes information necessary for building the interface for entering the input data.

5. Implementation details

The SIBIOS architecture follows the client-server model. A fat client approach has been adopted, which delegates some of the processing capabilities to the client in order to improve performance. The construction of the workflow accomplished by the *Workflow Builder*, for instance, resides on the client side since it is a graphically-oriented task which requires user's intervention. This approach reduces the communication overhead between the clients and the server.

From the user point of view, SIBIOS is a web-enabled application, which allows the user to launch the software from a web browser by using the Java Web Start technology [22]. The system does not suffer from the security restrictions associated with an applet or the interactivity restrictions associated with JavaScript [23] embedded web page.

SIBIOS adopts a distributed architecture where web service technologies are used for communications among components [8]. This feature increases the modularity of the system at the component level and promotes the utilization of standard technologies such as SOAP [9], WSDL [12], and UDDI [10]. The Glue [25] software is used to implement the communication among the components of SIBIOS, which can themselves be regarded as web services.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<TOP>
<SERVICE_APPLICATION>
  <SERVICE_NAME>fasta_nucleotide_service</SERVICE_NAME>
  <PATH_PAGE>
    <PAGE_CHARACTER> {title}job running:</PAGE_CHARACTER>
    <EXTRACT_LINK>P_uh='job running: ' t='{/title}{/head}' la='running:'ra=' {/title}' l='' r= ''</EXTRACT_LINK>
  </PATH_PAGE>
  …
</SERVICE_APPLICATION>
<URL_STRING>
    <QUERY_WITH_MULTIPLE_PARAMETERS>
      <BASIC_URL>http://www.ebi.ac.uk/cgi-bin/fasta/submit</BASIC_URL>
      <DEFAULT_PARAMETERS>tool=fasta!srchtype=interactive!program=fasta3!dbtype=DNA!matrix=none!stype=
          DNA/RNA</DEFAULT_PARAMETERS>
      <SEARCH_TITLE>title=XXXX</SEARCH_TITLE>
      <DATA_STORE>database=XXXX</DATA_STORE>
      <GAP_OPEN>gapopen=XXXX</GAP_OPEN>
      <GAP_EXTENSION>gapext=XXXX</GAP_EXTENSION>
      …
    </ QUERY_WITH_MULTIPLE_PARAMETERS>
</URL_STRING>
<EXTRACTION_RULES>
  <SUMMARY_OUTPUT>gh='opt bits' t='{a name' s=''
    <DATA_STORE>sh='' t='{' l='{/a}' r=''</DATA_STORE>
    <IDENTIFIER>sh=':{a href=' t='{/a}' l='}' r=''</IDENTIFIER>
    …
  </ SUMMARY_OUTPUT>
  <ALIGNMENT_DETAILS>
      gh='{a name=' t=' ' s=' {a name'
      <IDENTIFIER>sh=':{a href=' t='{/a}' l='}' r=''</IDENTIFIER>
      <ALIGNMENT_DESCRIPTION>sh=':{a href=' t=' ' l=' ' r=''</ALIGNMENT_DESCRIPTION>
      <ALIGNMENT_DATA>sh=':{a href=' t='' l=' ' r='_save</ALIGNMENT_DATA>
  </ALIGNMENT_DETAILS>
</EXTRACTION_RULES>
<INTERFACE_PARAMETER>
   <NAME>DATA_STORE</NAME>
   <DESCRIPTION>Choose Nucleic Acid Database</DESCRIPTION>
   <TYPE>
    <MULTIPLE_CHOICE>
    <CHOICE>
      <VALUE>emfun</VALUE>
      <DESCRIPTION>FUNGI</DESCRIPTION>
      …
</ INTERFACE_PARAMETER>
</TOP>
```

Figure 5. Extracts from FASTA service schema.

5.1. Workflow builder

The role of the workflow builder is to support the users in building the workflow and if necessary in modifying it during execution. It implements all the workflow capabilities described in Section 4. The result of the workflow specification is a workflow diagram. An example is shown in figure 6. Six services are connected by edges; some of them have filters (as indicated by small square boxes on the edges). When results from multiple services are input into a service, a connector (i.e., rectangular boxes) is added to combine the data using a *union* or a *join* operation. The second GenBank [19] service instance in this example is iterative, as shown by the iterative edge and the termination condition (i.e., also a small square box). The workflow diagram is stored on the client side in case the user may manipulate it later.

To add a new service to the workflow, the workflow builder combines the current workflow information (figure 7(a)) together with the user service specification (figure 7(b)) to submit a query to the service discovery component [21]. The list of matching services is displayed to the user and the selected service is added to the workflow diagram.

Once a service is added to the workflow, its specification is fetched from service schema wrapper. The specification lists the inputs, outputs, and environment parameters, as well as the other properties of a given service, and whether it can take multiple inputs. This information is used to generate a customized query interface for the service (figure 8(a)). In the interface, the values of most input and environment parameters have been provided based on service specification. Users are allowed to modify the environment parameter values (e.g., *gap_penalty* for BLASTP service), or to specify the input expression based on the antecedent services.

Before execution, the workflow diagram is converted into an XML file and submitted to the server. The workflow builder periodically queries the workflow web services to obtain the current execution status, and reflects it on the workflow diagram. Intermediate results are also fetched, thus, allowing

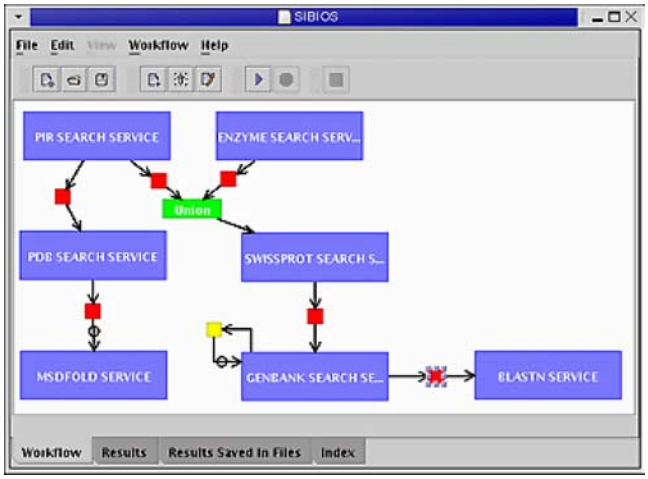


Figure 6. Workflow diagram example.

the user to inspect partial result before the entire workflow execution is completed (figure 8(b)).

5.2. Task engine

The role of the task engine is to execute the workflow based on its internal representation called *workflow object*. Workflow objects contain the execution dependency information (e.g. the set of services that immediately precede a given service in the workflow execution) so that multiple services within one workflow can be executed in parallel. Moreover, new instances of the workflow object and the task engine are created for each user session; therefore multi-user sessions can be executed concurrently.

The task engine (figure 9) executes the workflow by launching multiple services (tasks) in parallel where each service is associated with a task unit. The task unit is spawned by a centralized coordinating component, the *task coordinator*. The task coordinator is created as the main thread of the task engine and it works as the master process responsible for creating slave threads (task units), coordinating and allocating the jobs to the task units, maintaining the message passing, and managing the shared data objects. It performs in the following steps: (1) accommodate the workflow object and dependencies from the workflow manager; (2) schedule and trigger the execution of the tasks by acknowledging the dependencies; (3) create, communicate with the task unit through the task communicator; and (4) terminate or resume the execution of the workflow in case of user intervention.

The task unit is the actual service executor. As shown in figure 10, there are six sequential steps in each task unit: data filtering, data combination, URL construction, web service invocation, result parsing and result uploading.

Result filtering performs the data filtering based on the filters defined in the edge (flow control) and input expressions associated with the services. Once this step is completed, *Data combination* is performed when a service has more than one preceding service. The results of antecedent services are combined either by Join or Union connector. In the *URL construction* step, the URL of the web service is build, based on the input expression, the result data generated from the first two steps, and the service information (e.g., basic URl, default parameters, etc.) defined in the service schema.

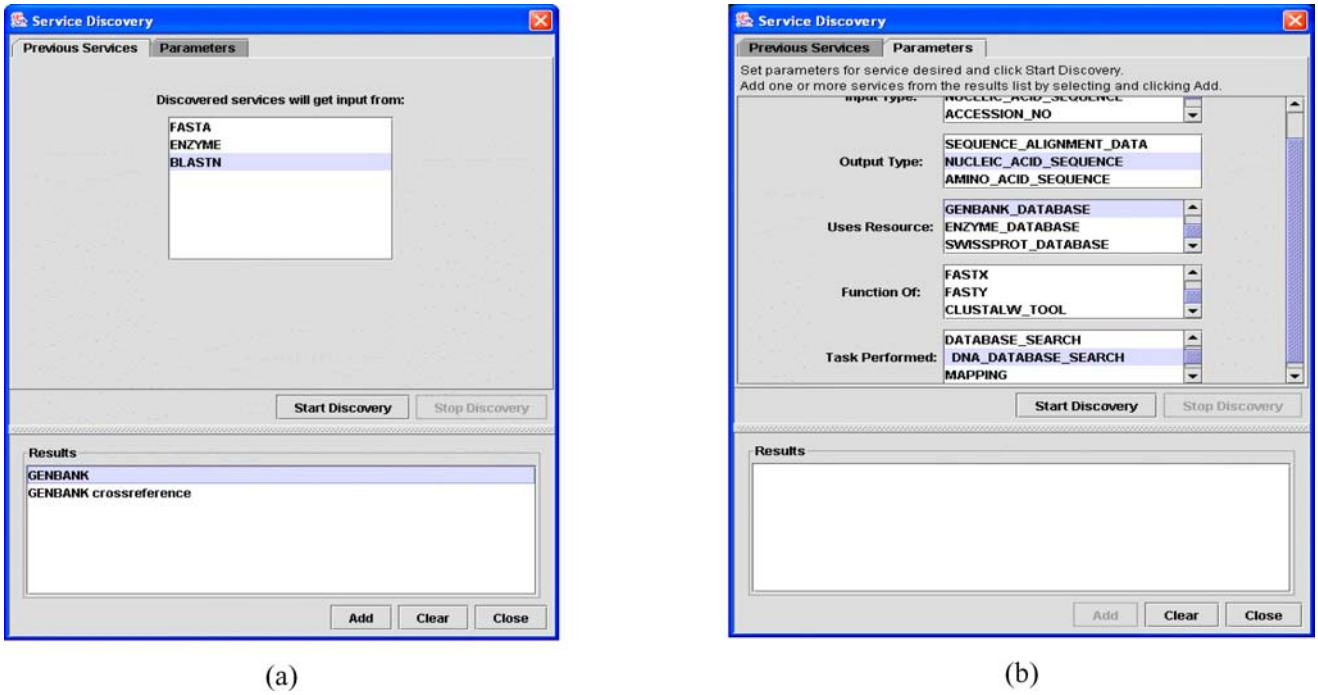
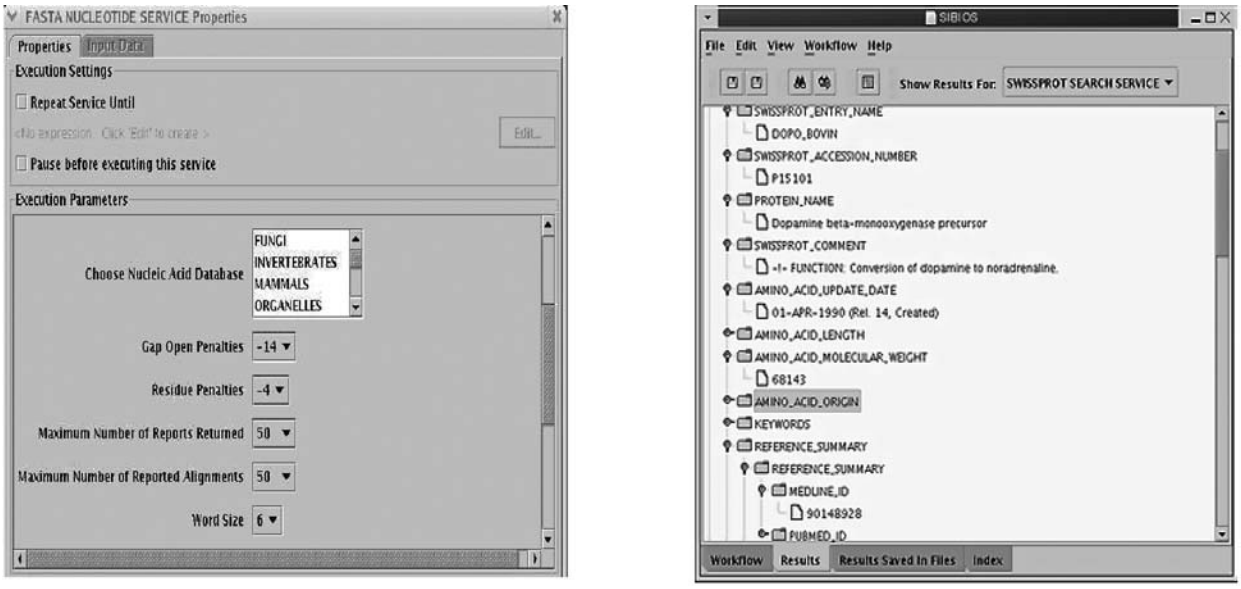


Figure 7. Service discovery interface.



(a) Default values from FASTA Nucleotide service

(b) Tree-viewed representation for partial results

Figure 8. Workflow building interface for (a) input data entry and (b) output presentation.

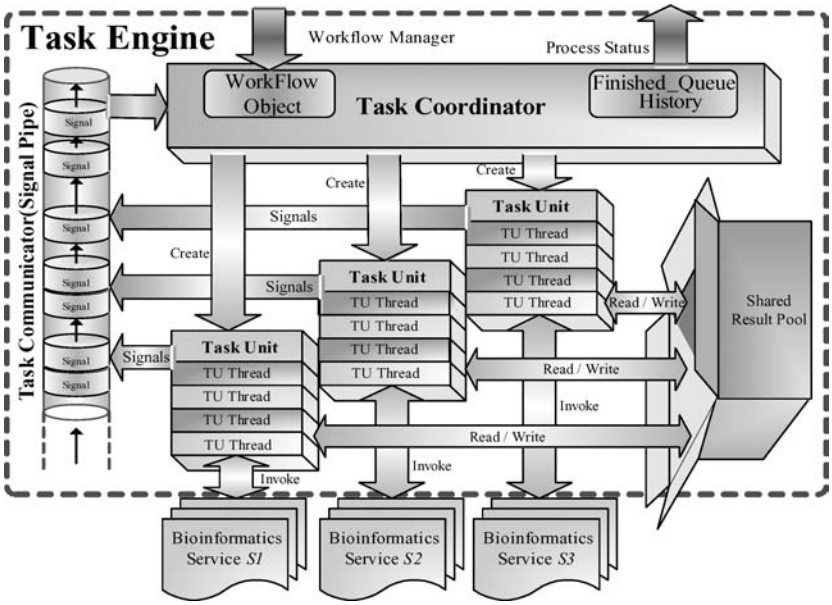


Figure 9. Global view of task engine architecture.

Next step is the *Web service invocation*, where the web service is invoked by using the URL built in the previous step. *Result parsing* refers to the extraction of information from the result pages by applying the extraction rules defined in the service schema (i.e., EXTRACTION_RULES shown in figure 5).

Parallelism within a given task unit occurs when more than one input need to be sent to the corresponding web service, thus a number of threads are forked and each thread is in charge of one input instance. The threads are running in parallel and are independent of each other. When the task unit finishes the execution, it sends a termination signal to the task coordinator through the task communicator channel. The task unit will (1) terminate normally if all the steps are executed successfully, (2) terminate early if the processing of the first two steps (data filtering and data combination) does not produce any result, or (3) terminate with failure if the service invocation is not successful (e.g., web service unavailable). Under a normal termination, two types of termination signals can be produced. The first signal indicates that the termination condition for an iterative service was not satisfied and therefore user intervention is necessary. In this case, a new task unit will be created after user intervention. The second type of signal is reserved for the case where the execution of the service is completed regardless of whether the service is an iterative service or not.

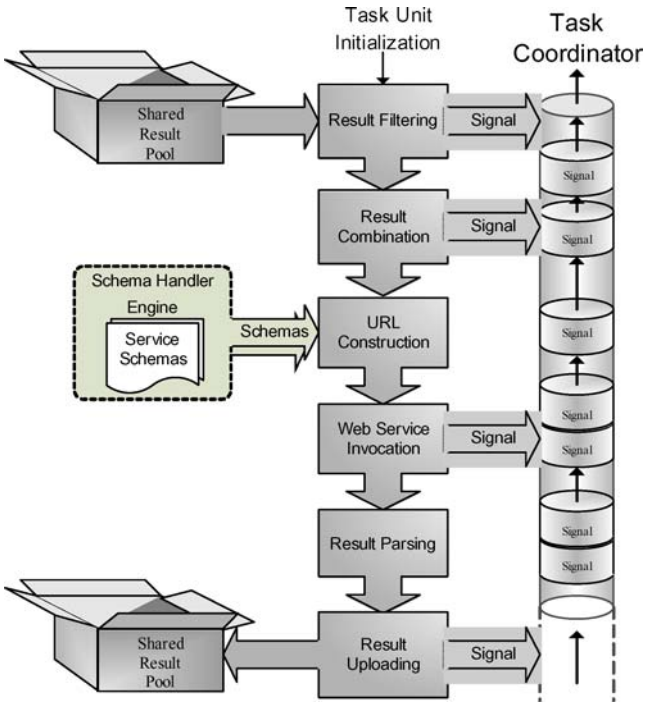


Figure 10. Task unit execution steps.

The task communicator is the communication media between the task coordinator and the task unit, which is indicated by the message pipe shown in figure 9. It is constructed following a producer-consumer model. The task unit is the signal producer which pushes the messages onto the pipe from one end and the task coordinator is the message consumer which receives the messages from the other end of the message pipe. The task communicator eliminates the need for maintaining a unique communication channel between each task unit and the task coordinator.

The shared result pool is a memory object that is used to support the communication between task units. It stores the results generated by one task unit for later retrieval by another task unit. Both the shared result pool and the message pipe provide a shared memory communication paradigm between the task coordinator (master) and the task units (slaves).

The results returned from the execution of remote services are classified and integrated under corresponding result types and stored in the internal data structure for the system to generate result trees for the user interface, further result manipulation and disk file storage. Figure 8(b) shows a sample UI representation in tree view of the results returned from Swiss-Prot database search.

6. Related work

The integration of bioinformatics tools is the focus of several ongoing academic and industrial projects [1,4,13,16,17,21,37, 42,44,46]. Early approaches [4,13,16,25,37] focused on the integration of databases. New approaches have been tackling the more general problem of service integration [14,17,21,35,38,44,45]. Some of these approaches, such as SRS, augmented database integration systems to include software tools. In addition to its database integration component, SRS allows the user to perform sequence analysis tasks using tools such as BLAST. However, the system is not workflow oriented. The system uses the results of each step to determine possible services for the next step. Also, the system assumes that the first step of any session is always a database search.

Other integration systems departed from considering the software tools as an add-on to database integration. For example, the approach adopted by ISYS [38] allows the user to execute services in a pipeline fashion. In ISYS, the user can execute a service, select a subset of the returned results, and the system will automatically determine the list of services that can be executed next. However, ISYS does not allow the user to specify all the steps in a workflow a priori. ISYS integrates both standalone and web services. It also includes representative services from various classes, such as sequence analysis, and database searches. In addition it supports free text searches using the Google [11] search tool.

The remaining and more recent service integration systems can be classified under two main categories. The objective of the systems under the first category is to develop systems with workflow capabilities. Examples of such systems include VIBE [21], TurboBench [42], ubertool [44], and Kepler [1]. Some of these systems impose restrictions on the types of services that can be integrated. They also support filtering [42,21] and organize the classes of services in a hierarchy which is based on functional taxonomy [44]. The majority of these systems lack support for dynamic workflows. Kepler system supports limited user intervention during workflow execution.

The second category of service integration systems includes Biomoby [45], and MyGrid [27]. The objective of these approaches is to define standards similar to those defined in the business field such as WSDL and UDDI that can facilitate the search, location, invocation, and result communication of services. MyGrid relies on a service specification, which is an extension of the WSDL specification of services. It also assumes a registration mechanism similar to the UDDI registration. To add new services to the system, service providers are expected to register their service through a registry that is accessed by the integration system. The registry information includes the service name, its URL, and the description of service using a service description template. Support for workflow capabilities in the case of the service integration systems in this category is limited when compared to the integration systems of the first category.

SIBIOS combines the features of the two service integration categories by supporting enhanced workflow capabilities and relying on ontologies for service specification. The main difference between SIBIOS and workflow based-systems such as VIBE, for example, is the support for dynamic workflows. In Kepler system, the workflow generated by the user is an abstract workflow that includes the task specifications (e.g. homology search) instead of the exact service names (e.g.

BLastn). In this sense, SIBIOS provides the user with more control on the selection of the services that can accomplish a bioinformatics task. The main difference between SIBIOS and MyGrid is that SIBIOS does not rely on explicit service registration by service provider. This aspect renders the service execution and service results extraction more difficult because it has to rely on wrapper-based techniques.

7. Conclusion

This paper presented SIBIOS, a new integration system for bioinformatics services. The challenges that need to be addressed to support the integration of distributed and heterogeneous bioinformatics services have also been described in this paper. SIBIOS supports a new dynamic service workflow model that allows the specification of several features rendering automatic execution of bioinformatics workflow possible while still allowing the user to intervene during workflow execution. The prototype of SIBIOS is available at http://sibios.engr.iupui.edu/. User can download SIBIOS client from this link. Future research will focus on improving the system performance, fault tolerance as well as supporting advanced task scheduling and incorporating more sophisticated operators in the workflow.

Acknowledgment

This project is supported in part by NSF CAREER DBI-DBI-0133946 and NSF DBI-0110854.

References

[1] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher and S. Mock, Kepler: An extensible system for design and execution of scientific workflows, in *16th Intl. Conference on Scientific and Statistical Database Management (SSDBM)* (Santorini Island, Greece, 2004).

[2] S.F. Altschul, T.L. Madden, A.A. Schaffer, J. Zhang, Z. Zhang, W. Miller and D.J. Lipman, Gapped BLAST and PSI-BLAST: A new generation of protein database search programs, Nucleic Acids Res 25(17) (1997) 3389–3402.

[3] BLOCKS, J.G. Henikoff, E.A. Greene, S. Pietrokovski and S. Henikoff, Increased coverage of protein families with the blocks database servers, Nucl. Acids Res. 28 (2000) 228–230.

[4] Z. Ben Miled, N. Li, G. Kellett, B. Sipes and O. Bukhres, Complex life science multidatabase queries, in: *Proceedings of the IEEE*, vol. 90, no. 11, (2002).

[5] D. Buttler, M. Coleman, T. Critchlow, R. Fileto, W. Han, C. Pu, D. Rocco and L. Xiong, Querying multiple bioinformatics information sources: Can semantic web research help?, SIGMOD Record 31(4) (2002).

[6] A. Bairoch, The ENZYME database in 2000, Nucleic Acids Res. 28 (2000) 304–305.

[7] T. Berners-Lee, J. Hendler and O. Lassila, The semantic web, Scientific American (2001).

[8] D. Booth, M. Champion, C. Ferris, F. McCabe, E. Newcomer and D. Orchard, Web services architecture, *W3C Working Draft* (2003).

[9] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte and M.D. Winer, Simple object access protocol (SOAP) 1.1, *W3C Note* (2000).

[10] T. Bellwood et al., UDDI Spec, Technical Committee Specification, (2002).

[11] S. Brin and L. Page, The anatomy of a large scale hypertextual web search engine, *7th WWW Conference*, (1998).

[12] E. Christensen, F. Curbera, G. Meredith and S. Weerawarana, Web services description language (WSDL) 1.1, W3C Note (2001).

[13] S.B. Davidson, O.P. Buneman, J. Crabtree, V. Tannen, G.C. Overton and L. Wong, BioKleisli: Integrating biomedical data and analysis packages, in: *Bioinformatics: Databases and Systems*, S. Letovsky (ed.), Kluwer Academic Publishers, Norwell, MA pp. 201–211 (1999).

[14] DoubleTwist, Inc., http://www.doubletwist.com

[15] eMOTIF, J.Y. Huang and D.L. brutlag, The EMOTIF database, Nucleic Acid Res., 21(1) (2000) 202–204.

[16] T. Etzold, A. Ulyanov and P. Argos, SRS: Information retrieval system for molecular biology data banks, Methods Enzymol 266 (1996) 114–128.

[17] Entigen Corporation (eBioinformatics, Inc., and Empatheon, Inc.), http://www.entigen.com/

[18] Entrez, Entrez's 3D-structure database, Nucl. Acids. Res. 31 (2003) 474–477.

[19] GenBank, GenBank, Nucl. Acids. Res. 31 (2003) 23–27.

[20] Genome resources and searches, http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db = Genome

[21] INCOGEN, Inc., VIBE: Visual integrated bioinformatics, white paper, http://www.incogen.com

[22] Java Web Start, http://java.sun.com/products/javawebstart/

[23] JavaScript, http://wp.netscape.com/eng/mozilla/3.0/handbook/javascript/

[24] K. Kochut and J. Arnold, et al., IntelliGEN: A distributed workflow system for discovering protein-protein interactions, *Distributed and Parallel Databases* 13 (2003) 43–72.

[25] M. Kanehisa, S. Goto, S. Kawashima, Y. Okuno and M. Hattori, The KEGG resource for deciphering the genome, Nucl. Acids. Res. 32 (2004) D277–D280.

[26] L. Moreau, S. Miles, C. Goble, M. Greenwood, V. Dialani, M. Addis, N. Alpdemir, R. Cawley, D. De Roure, J. Ferris, R. Gaizauskas, K. Glover, C. Greenhalgh, M. Greenwood, P. Li, X. Liu, P. Lord, M. Luck, D. Marvin, T. Oinn, N. Paton, S. Pettifer, M.V Radenkovic, A. Roberts, A. Robinson, T. Rodden, M. Senger, N. Sharman, R. Stevens, B. Warboys, A. Wipat and C. Wroe, On the use of agents in a bioinformatics grid, in: *Proceedings of the Third IEEE/ACM CCGRID'2003 Workshop on Agent Based Cluster and Grid Computing*, Sangsan Lee, Satoshi Sekguchi, Satoshi Matsuoka, and Mitsuhisa Sato (eds.), Tokyo, Japan, (2003) pp. 653–661.

[27] L. Moreau, S. Miles, C. Goble, M. Greenwood, V. Dialani, M. Addis, N. Alpdemir, R. Cawley, D. De Roure, J. Ferris, R. Gaizauskas, K. Glover, C. Greenhalgh, M. Greenwood, P. Li, X. Liu, P. Lord, M. Luck, D. Marvin, T. Oinn, N. Paton, S. Pettifer, M. V Radenkovic, A. Roberts, A. Robinson, T. Rodden, M. Senger, N. Sharman, R. Stevens, B. Warboys, A. Wipat and C. Wroe, On the Use of Agents in a bioInformatics grid, in: *Proceedings of the Third IEEE/ACM CCGRID'2003 Workshop on Agent Based Cluster and Grid Computing*, Sangsan Lee, Satoshi Sekguchi, Satoshi Matsuoka, and Mitsuhisa Sato (eds.), Tokyo, Japan, (2003) pp. 653–661.

[28] OWL, A non-redundant composite protein sequence database, Nucl. Acids. Res. 22 (1994) 3574–3577.

[29] Protein Sequence Analysis, a practical guide. http://www.bioinf.man.ac.uk/dbbrowser/bioactivity/

[30] PIR, The protein information resource (PIR), Nucl. Acids. Res. 28 (2000) 41–44.

[31] PROSITE, The PROSITE database, Nucl. Acids. Res. 30 (2002) 235–238.

[32] Profiles, http://hits.isb-sib.ch/cgi-bin/PFSCAN.

[33] Pfam, The Pfam protein families database, Nucl. Acids. Res. 32 (2004) D138–D141.

[34] W.R. Pearson and D.J. Lipman, improved tools for biological sequence comparison, PNAS 85 (1988) 2444–2448, W.R. Pearson, Rapid and

sensitive sequence comparison with FASTP and FASTA, Methods in Enzymology 183 (1990) 63–98.
[35] P. Rice, I. Longden and A. Bleasby, EMBOSS: The European molecular biology open software suite, Trends in Genetics, 16(6) (2000) 276–277.
[36] D. Rocco and T. Critchlow, Discovery and Classification of Bioinformatics Web Services, *Lawrence Livermore National Laboratory Technical Report.* UCRL-JC-149963 (2002).
[37] 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) (2000) 184–186.
[38] A. Siepel, A. Tolopko, A. Farmer, P. Steadman, F. Schilkey, B.D. Perry and W. Beavis, An integration platform for heterogeneous bioinformatics software components, IBM Systems Journal 40(2) 570–591.
[39] Swiss-Prot, The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003, Nucl. Acids. Res. 31 (2003) 365–370.
[40] S. Schulze-Kremer, Ontologies for molecular biology, Third Pacific Symposium on Biocomputing (1998) 695–706.
[41] Transeq, EMBOSS tool for translating DNA/RNA into protein. http://www.ebi.ac.uk/emboss/transeq/
[42] TurboWorx™, http://www.turboworx.com
[43] The workflow portal, The Workflow Handbook 2004, Published in association with the *Workflow Management Coalition* (WfMC), Layna Fischer (ed.).
[44] Ubertool, http://www.science-factory.com/products.html
[45] M.D. Wilkinson and M. Links, BioMOBY: An open-source biological web services proposal, Briefings in Bioinformatics 3(4) (2002) 331–341.
[46] GCG® Wisconsin Package™, http://www.accelrys.com/products/seqweb
[47] C. Wroe, R. Stevens, C. Goble, A. Boberts and M. Greenwood, A suite of DAML + OIL ontologies to describe bioinformatics web services and data, International Journal of Cooperative Information Systems 12(2) (2003).

E-mail: ngao@iupui.edu

E-mail: niali@iupui.edu

E-mail: jwchen@iupui.edu

E-mail: mmahoui@iupui.edu

E-mail: bukhres@iupui.edu

E-mail: linglu@iupui.edu

E-mail: zmiled@iupui.edu