AN AGENT SYSTEM UPON SEMANTIC WEB TECHNOLOGIES TO PROVIDE A FUNGAL GENOMICS DATA WAREHOUSE

Farzad Kohantorabi

A thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Computer Science

Concordia University

Montréal, Québec, Canada

September 2006

Your file Votre référence
ISBN: 978-0-494-20776-5
Our file Notre référence
ISBN: 978-0-494-20776-5

Abstract

An Agent System upon Semantic Web Technologies to Provide a Fungal Genomics Data Warehouse

Farzad Kohantorabi

Biologists and bioinformaticians are accustomed to going through analysis steps in which they employ multiple data sources such as protein sequence and protein interaction databases. However, biological sources have a distributed, dynamic, and heterogeneous nature and finding the right source of data is not an easy task. Therefore, the need for integrated biological data sources has received a fair amount of attention from both computer and biology communities in the recent years.

In addition to that, access to biological data and service sources has become an issue in the recent years due to the use of inconsistent terminologies. That is, biological vocabulary is diverse in nature, and people do not use the same terminology to describe their data and services. Therefore, finding the right service and interpreting data received from multiple data sources is an issue.

In this research, we build a data warehouse containing integrated fungal genomics data for multiple fungi species from multiple sources, a data schema for the integrated data, transformers for changing the data from external sources to the integrated data schema, and an agent based ETL system for the data warehouse.

Acknowledgments

No one successfully finishes his mission alone and I am not an exception. I would like to acknowledge my supervisor, Dr. Gregory Butler, for his genuine knowledge, timely support, and exceptional scientific vision and understanding. Without him this work would never see the light of the day.

In addition, I would like to acknowledge my family for what they have been in my life. My parents put a tiredless effort in bringing me up and showing me the way of life, and my brothers never left me alone in the hard days and they always helped me move forward. Without my family I would never be the person I am now.

Finally, I would like to thank my friends who supported me in all the stages of my life. Specially A Arasteh who helped me in this thesis by taking the time and effort to proofread this text. In addition to him, A Shaneh, B Zamani, M Behbahani, K Hamidya, P Farzadmanesh, A Abarahamian, A Avanesh, N Javaher, M Fatahian, I Rajaee, M Zolmajd and his lovely wife is a short list of their names. I treasure your friendship.

Thank you all

Farzad Kohantorabi

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computational data has become a primary source of new biological insights, and bioinformatic software now contributes significantly to biological initiatives. Nowadays, a daily life of a bioinformatician typically starts by querying biological databases for annotations, sequence similarities and so on, and it continues by examining the results and inferring new findings.

Baker et al. [BBB+98] explain that when a biologist needs to ask questions, he takes certain steps to find the answers. First, he needs to find which sources might have the relevant information. He continues by examining the content of the sources to find whether not they are good enough to answer the questions. Next is to find the proper way of querying each source. After that is the communication with the selected sources. Then after, he needs to interpret the format of the returned result. At the same time, he may also need to ask complex questions. And finally, he needs to integrate the results if they are from multiple sources.

The above mentioned steps are the rough generalization of the reality; however, they are applicable to most scenarios. As an example, Shaneh et al. [SB06] are interested in predicting the Glycosylation sites [dfG] in certain proteins, and they are using improved neural network techniques for this purpose. It's needless to say their

neural network needs proper training data.

A google search for protein databases will list quite a few protein databases that are a good candidate for building their training data. InterPro [AAB+01], UniProt [BAW+05], Protein Data Bank (PDB) [BWF+00], the Human Protein Reference Database (HPRD) [PNA+03] are examples of such databases. A closer study of these databases show that InterPro and PDB are perfect matches for building the training data.

ID/Accession	Protein Name	Length	Organism Name	Taxon Group	UniRef90/50	Matched Fields
GALT5_RAT / O88422	Polypeptide N-acetylgalactosaminyltransferase 5	930	Rattus norvegicus	Eukfmammal	UniRef90_O88422 UniRef50_Q7Z7B9	Comment=>Muc
LON_ECOLI / P0A9M0	ATP-dependent protease La	784	Escherichia coli	EacGamma-proteo	UniRef90_P46067 UniRef50_P46067	Gene Name=>muc
MUC1_HUMAN / P15941	Mucin-1 precursor	1255	Homo sapiens	Eukfmammal	UniRef90_P15941 UniRef50_P15941	Protein Name=>MUC
MUC1_HYLLA / Q29435	Mucin-1 precursor	475	Hylobates lar	Eukfmammal	UniRef90_Q29435 UniRef50_P15941	Protein Name=>MUC

Figure 1: UniProt search result screen

Therefore, what Shaneh et al. need to do is to search for the selected proteins in the InterPro database. For example, imagine one of the proteins is Mucin. A search in the UniProt web site will result in the Figure 1[1], in which a list of proteins related to Mucin is shown. Lets say that the MUC1_HUMAN protein is a good candidate for the training data. A click on the MUC1_HUMAN link navigates to the description page of the protein, in which a description of regions in the protein is provided. Table 1 shows the relevant Glycosylation site data from the description page. Table 1 shows the beginning position and the end position of the site. Therefore, we will need the sequence of the protein to find out the exact sequence of the protein, and eventually the data for the neural network is ready. (For the sake of simplicity it is assumed that only the window around the Glycosylation site is enough for the neural network. The original work may be consulted for exact details of the algorithm and the feeds to the neural network).

[1]The picture has been modified from its original form to better fit in this page.

Feature	Description	Begin	End
GLYCOSYLATION SITE	N-linked (GlcNAc...) (POTENTIAL)	957	957
GLYCOSYLATION SITE	N-linked (GlcNAc...) (POTENTIAL)	975	975
GLYCOSYLATION SITE	N-linked (GlcNAc...) (POTENTIAL)	1029	1029
GLYCOSYLATION SITE	N-linked (GlcNAc...) (POTENTIAL)	1055	1055
GLYCOSYLATION SITE	N-linked (GlcNAc...) (POTENTIAL)	1133	1133

Table 1: Glycosylation sites for Mucin-1 human protein

As is in the case of UniProt and PDB, many popular biology databases provide websites where people submit their queries and construct the dataset they need for their research. There are even data warehouse systems that integrate data from multiple sources and provide them in an uniform system; therefore, they save the integration time for their users. Biozon [BY06b] is an example of such systems. However, these sources have been accommodated for human access, and they do not provide convenient interfaces for software programs.

In recent years, there have been relevant initiatives to fill this gap. For example, ToolBus [YENS05], EBI web services [PSK+05], DDBJ web services [MS00], NCBI eFetch [eFe], and BIND [BDW+01] offer services that computer programs can use to retrieve biological data and perform their computation. However, these services are tightly coupled to the definition of their data, and locating and using the right service is still a problem.

There is an ongoing debate in the life sciences community on the use of multi-agent systems in the field of life sciences. Burger [Bur] discusses that there are firm evidences that multi-agent systems have something to offer that other systems do not. He points out that, specifically in the field of Semantic Web, all the significant work in the scientific community has been done on multi-agent systems, and the Semantic Web is a proven candidate to solve some of the existing problems in the area.

The problem of distributed and heterogeneous data sources is a classical data integration problem that has been addressed by three types of programs: portal oriented systems, mediator oriented systems, and data warehouse oriented systems.

In this research, a fungal genomics data warehouse is built for a rich set of fungi species. Furthermore, an integrated data schema for the data stored in the data warehouse is designed and used by a useful set of transformers that change the format of external data to the integrated data schema.

In addition to the data warehouse, an agent system that maintains the data warehouse and is aimed to provide access to the data warehouse in a distributed environment is introduced. The agent system is powered with ontologies for the sake of locating services according to their service concept and data concept, thus making the search for services more accurate and more successful.

The use of ontologies in the research involves the introduction of a service ontology and a data ontology that are both shared between service seekers and service providers. In the context of a data warehouse, as an integrated database, the service ontology contains generic services such as `retrieve`. A parameter that is passed to the service is a query involving the data ontology. In this case there is a separation between the service ontology and the data ontology. Alternatively, specific services to retrieve genes or proteins could be available. In this case, the service ontology (below the generic retrieve service) mirrors a subset of the data ontology.

The agent system in this research works as an ETL engine for the data warehouse. That is, it goes to the external sources and fetches the data, transforms the data according to its integrated data schema and using the developed transformer objects, and loads it to the data warehouse. The agent system is flexible in accepting new ETL agents that integrate new data types into the data warehouse.

The data warehouse is the first in its kind to build an integrated fungal genomics database from multiple species. The data warehouse has about 4 million records from 21 fungi species. New data is still added to the data warehouse, and it is expected the final size will be around 10 million records. In addition, the data warehouse is the first that stores multiple classifications of its data and the mapping between them.

1.1 FungalWeb Project

The first years of bioinformatics was spent on the storage issues, management, and analysis of biological data. However, bioinformatics has now entered into a new era in which a large volume of data is either manually or computationally being generated, diverse range of data types exists, and data access/manipulation has become an issue. After all, data sources are becoming more distributed, they are in different forms and formats, and new versions of data are out more frequently [BBH04].

As a result of the changes in bioinformatics, people are now becoming more concerned by the level of computer skills a biologist needs to acquire in order to benefit from the currently available biological knowledge sources. This concern brings into attention the need for systems that ease access to the biological knowledge. A proper candidate to provide technological enhancement in this respect is Semantic Web. Sematic Web is a recent challenge for using meta data level information to access knowledge across a collection of sources.

FungalWeb, "Ontology, the Semantic Web, Intelligent Systems for Fungal Genomics" is a research project at Concordia University, McGill University, and Université de Montréal that aims to bring the power of ontologies, machine learning, and natural language processing to the Fungal Genomics area.

In principal, FungalWeb is comprised of the following components:

- Ontology: Ontologies are now an integral part of knowledge base systems. Ontologies help humans and computers understand the knowledge obtained from different sources. FungalWeb introduces an ontology which feature fungal and enzyme specific concepts and data. More details of the FungalWeb ontology are provided in the rest of this thesis.

- Agent System: Accessing multiple sources of data to conduct complex queries is an issue in a distributed data space, which is the case in the biological data.

Agent systems have proven useful for such issues. The FungalWeb project comes with an agent system that wraps multiple sources of biological data and provides access to them. The main concern of this thesis is the agent system.

- Text Mining: A vast amount of scientific knowledge exists in the form of free text, most of it is in the scientific papers. Selecting the right textual source for acquiring the required knowledge is an issue when it has to be done by human forces. An automated computer system which is able to scan textual sources and recommend relevant ones will save a lot of time and increases quality in the process of knowledge acquisition. FungalWeb project employs computational linguistic approaches to analyze scientific texts and extract/index their knowledge.

- Relational Data Mining: Relational data mining is becoming a promising approach for situations where data mining has to span multiple sources. The nature of biological literature and data encourages using such approaching as the biological data is highly correlated, and inferring knowledge needs examining multiple sources at the same time. FungalWeb project employs probabilistic relation data mining to build regulatory networks, which helps with validating laboratory experiments.

The main concern of the FunalWeb project is fungal species. FungalWeb project is also affiliated with the Fungal Genomics project in the Concordia University. The Fungal Genomics project aims to find new genes for fungal species and the function of enzymes produced by those genes. In the long run, it is planned that an integrated system built on the top of the FungalWeb project's components will help the Fungal Genomics project.

The FungalWeb project was funded by Génome Québec, and it has been awarded

the second prize at the International Semantic Web Conference in the category of Semantic Web Challenges 2005. More informations of the FungalWeb project is available online at `http://www.cs.concordia.ca/FungalWeb/`.

Chapter 2

Tools and Background

This research depends upon the existence of specific tools. This chapter brings a brief introduction to those tools and to related research.

2.1 Ontology

In recent years managing the flow of information has become an issue because the information sources are numerously being made available, and the size of the available information is increasing at a fast pace. Moreover, the recent advances in the technology has eased the process of producing new information out of existing information, and it has introduced easer ways of sharing the new information (and the knowledge).

However, people tend to use different terminologies when they talk about things in the same domain. An example is the biology domain, in which people use totally different terminologies to refer to the same things. For example, a residue and an amino acid refer to the same thing in a protein sequence. Therefore, the need to agree on a consistent terminology is a must in order to enable information and knowledge sharing. Without such an agreement, it is possible that at some point people do not understand the domain knowledge that comes from other people.

An ontology [NM01] is a formal agreement on the terminology of a specific domain. People may develop an ontology to agree how the shared information should be read, reasoned, and interpreted. With the proper ontology it is understandable that the term residue and the term amino acid refer to the same thing in a protein sequence.

In addition, ontologies are not only to be used by humans. A significant benefit of ontologies is that they help computers understand the semantics of the information they take from external sources. For example, ontologies are a great improvement for agent systems where large amount of data from different sources is distributedly processed, and ontologies enhance business to business transactions in the e-Commerce systems.

2.1.1 What is an Ontology?

An ontology is a description of concepts (or interchangeably Classes) in a specific domain, properties (also referred to as slots, roles or attributes) of each concept, and restrictions of each property (also referred to as facets or role restrictions). In addition to this abstract definition, an ontology may have instances (also referred to as individuals) of each concept. All together, one may refer to an ontology as a knowledge base of a domain.

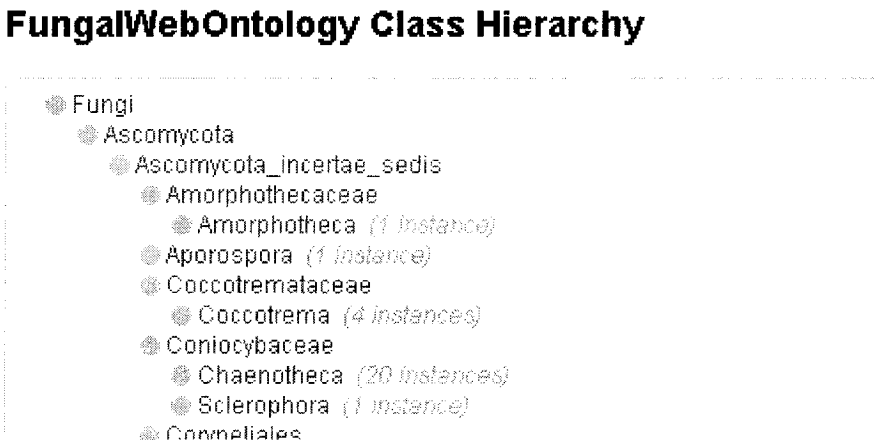


Figure 2: An example of a class hierarchy in an ontology

Classes in an ontology are the domain concepts. For example, `Fungi` is a general class for all fungi species. It is also possible to specialize a class by subclassing. Subclassing is the process of introducing new classes that break down a general class. For example, `Ascomycota` and `Microsporidia` are two subclasses of `Fungi`, which means they are both `Fungi` but they are different (the specialization of the `Fungi` concept). Figure 2 shows a portion of the FungalWeb ontology [SNBHB05], which will be discussed later in this chapter.

2.1.2 How to Design an Ontology?

There is no restriction on how to design an ontology, and it is totally up to the domain expert's design decision what the concepts should be and so on. However, Noy et al. [NM01] suggest the following steps:

- Determine the domain and scope of the ontology: During this step you limit the border of concepts you want to include in the ontology.
- Try to reuse existing ontologies: It is a good idea to integrate the currently available ontologies and add to them. As an instance, the FungalWeb ontology integrates the Gene Ontology [ABB+00] and TAMBIS ontology [BBB+98].
- List the terms you need in the ontology: Go through the things you may need in the ontology.
- Create your class hierarchy: This step is to find out what terms you want to introduce as a class and in what order you want to arrange them.
- Define the slots.
- Define the restrictions on the slots.
- Create instances if there is a need for any.

The fine points of each step above is out of the scope of this thesis, but [NM01] can be consulted for a complete description of each step.

Finally, in terms of software support for ontologies, there are some graphical user interfaces for developing ontologies. Protégé (`http://protege.stanford.edu/`) is a successful application that helps with the design and development of ontologies. In addition to the designers, there are other applications that facilitate use of ontologies. The RACER server [HM01] is an example of such an application, which will be discussed in more details in this thesis.

2.2 RACER

The advent of the Semantic Web has brought new challenges into the area of knowledge representation, and it has lead to introduction of new knowledge representation languages such as DAML+OIL [MFHS02] and OWL [HHH+03]. These languages have enough facilities to represent knowledge in a graph or tree manner. In addition, in order to add more power, some of these languages have been equipped with description logics so that large fragments of the language can be expressed [HM03]. OWL DL is an example of such a language.

RACER (Renamed A-Box and Concept Expression Reasoner) [HM01] is an inference engine for description logic languages, and it is an essential to add practical experience to the description knowledge languages. As a reasoner, the RACER supplies a query language that facilitates asking description logic queries.

From the implementation point of view, the RACER comes as a standalone server which is able to load DAML+OIL and OWL files and lets the users query the loaded knowledge in its query language. The communication between a user client and the RACER server is possible through the traditional HTTP and TCP protocols.

The RACER communication protocol is fairly straightforward and the proper language specific implementations are available online. For example, JRacer is the Java wrapper of the RACER server's protocol. A list of currently available RACER APIs is available online at `http://www.racer-systems.com/products/download/nativelibraries.phtml`.

The RACER server is also compatible with currently available description logic protocols, and it responds to the programs that talk in DIG protocol [BMC03]. The DIG protocol is a least common denominator description logic protocol of the existing protocols, and it enables applications to talk to description logic servers over HTTP.

Although DIG is a need as a standard and it covers all the standard description logic queries, it can not cover system specific queries. As a result, the RACER server supports an additional TCP protocol that is different from DIG, and it is more like the KRSS [PSS93]. This additional protocol enables spontaneous user interactions.

In addition to the protocols, there are applications that provide either shell or GUI access to the RACER server. Examples of such applications are RICE [MCH03], Ontoligent [BSBH06] and OntoXpl [HLS04].

The description of the inference types that the RACER server supports is beyond the scope of this thesis, and more details can be found in [HM03].

2.2.1 nRQL

RACER can be used to query concepts and individuals of an ontology. However, the RACER query language has limited support for ontology individuals and it supports concepts more. The new RACER Query Language (nRQL) [HMW04] is an extension of the RACER query language with the support of querying individuals.

The nRQL query language modifies and adds to RACER querying facilities for individuals. For example, one can use variables inside nRQL queries to retrieve a list of individuals or form complex queries on them. Details of nRQL specification can

be found in [HMW04]‘.

The combination of RACER, nRQL, and a simple GUI gives users enough facilities to ask their questions and expect an answer in a reasonable time. Figure 3 shows an example of this combination for the FungalWeb ontology. FungalWeb ontology is a unique ontology in the area of fungal genomics. Section 2.3 discusses the FungalWeb ontology in more details.

Figure 3 shows how the RACER server answers nRQL queries of the FungalWeb ontology.

2.3 FungalWeb Ontology

Fungi are now increasingly used in industry. However, the appropriate support for such industrial usage is still in its early stages, and it needs further developments. Baker et al. [BWSN+05] mention that many decisions in R&D teams are made on the basis of incomplete knowledge. To fill this gap, they suggest that a range of interdisciplinary ontologies should be introduced. The suggested range covers taxonomy, gene discovery, protein family classification, enzyme characterization, enzyme improvement, enzyme production, enzyme substrates, enzyme performance benchmarking, and market niche [BWSN+05].

The FungalWeb ontoloy is an example of such an ontology, which has been developed in the Computer Science and Software Engineeringdepartment of the Concordia University as part of the FungalWeb project.

The FungalWeb ontology [SNBHB05] is the integration of the available ontologies, the available databases, the domain expert knowledge, and the data obtained from various web sites. The resources for the FungalWeb ontology include [BWSN+05]: taxonomy databases such as NCBI taxonomy [WBB+06] and NEWT [PPFB03], NCBI literature databases, enzyme databases such as BRENDA

```
1- Give me all the instances of Neolectaceae?
retrieve (?x) (?x |http://a.com/ontology#Neolectaceae|)

Answer: (((?x |http://a.com/ontology#Neolecta_irregularis|))
         ((?x |http://a.com/ontology#Neolecta_vitellina|)))

2- All Agaricales have been reported to have enzyme Laccase?
retrieve (?x) (AND (?x |http://a.com/ontology#Agaricales|)
                   (?x |http://a.com/ontology#Laccase|
         |http://a.com/ontology#Has_been_reported_to_have_enzyme|))

Answer: (((?x |http://a.com/ontology#Lentinus|))
         ((?x |http://a.com/ontology#Coprinopsis_cinerea|))
         ((?x |http://a.com/ontology#Schizophyllum_commune|))
         ((?x |http://a.com/ontology#Agaricus_bisporus|))
         ((?x |http://a.com/ontology#Coprinus_cinereus|))
         ((?x |http://a.com/ontology#Pleurotus_sajor-caju|))
         ((?x |http://a.com/ontology#Lactarius_paperatus|))
         ((?x |http://a.com/ontology#Marasmius_quercophilus|)))

3- Which enzymes are being used in baking and brewing?
retrieve (?x) (AND (AND (?x |http://a.com/ontology#Enzyme|)
                        (?x |http://a.com/ontology#Baking|
                            |http://a.com/ontology#Is_being_used_in|))
                        (?x |http://a.com/ontology#Brewing|
                            |http://a.com/ontology#Is_being_used_in|))

Answer: (((?x |http://a.com/ontology#Protease|)))

4- All enzymes have been reported to be found in Neurospora Crassa?
retieve (?x) (AND (?x |http://a.com/ontology#Enzyme|)
        (?x |http://a.com/ontology#Neurospora_crassa|
            |http://a.com/ontology#Has_been_reported_to_be_found_in|))

Answer: (((?x |http://a.com/ontology#Xylanase|))
         ((?x |http://a.com/ontology#Cellulase|))
         ((?x |http://a.com/ontology#Pectinase|))
         ((?x |http://a.com/ontology#Lipase|))
         ((?x |http://a.com/ontology#Laccase|)))
```

Figure 3: nRQL examples from the FungalWeb ontology [SNBHB05]

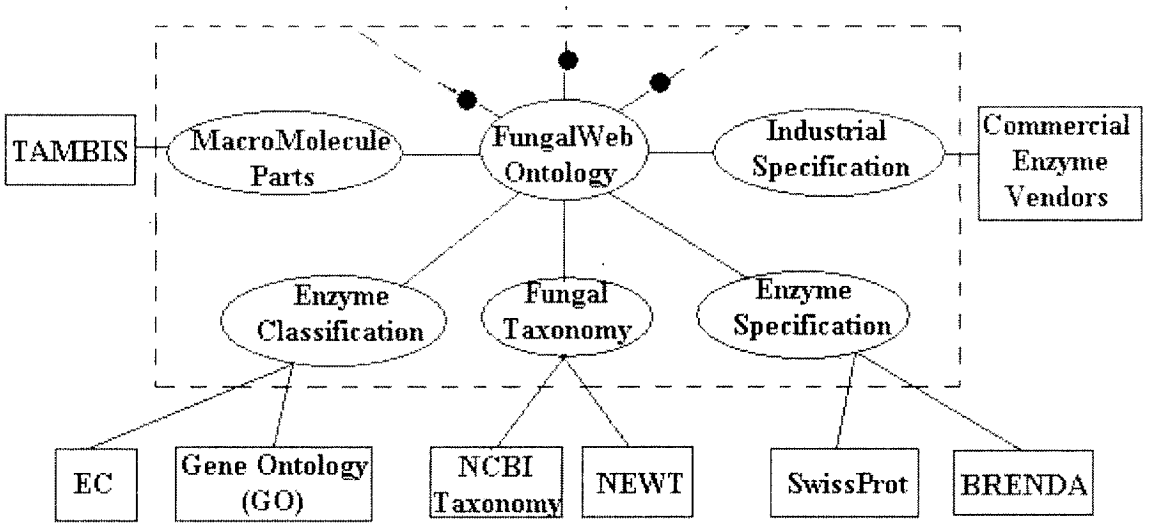


Figure 4: FungalWeb ontology sources [SNBHB05]

Source Name	Instances	Concepts
BRENDA & SwissProt	105	35
EC & GO	1296	198
NCBI Taxonomy Database	10870	3340
TAMBIS	0	22
Commercial Enzyme Vendors	401	6

Table 2: The size of data obtained from various sources [SNBHB05]

enzyme database [SCE+04] and Enzyme Nomenclature Database, Saccharomyces Genome Database (SGD) [HBC+], Neurospora Crassa Genome Database [Neu], the available information from commercial enzymes [Bai00], and the existing bio-ontologies such as Gene Ontology [ABB+00] and TAMBIS [BBB+98]. The integration of the existing ontology involves merging, mapping, and partially copying instance data from those ontologies. Figure 4 shows the existing sources of the FungalWeb ontology.

The integration process of the FungalWeb ontology has been manual, during which the data (ontology instances) has been transformed into FungalWeb specific concept schema and the concepts have been carefully unified. As of November 2005, the size of the ontology is 3667 concepts and 12686 instance, for which the details are in Table 2.

More details on the FungalWeb ontology can be found in Shaban-Nejad's master's

thesis [SN05].

2.4 DECAF

DECAF (Distributed, Environment Centered Agent Framework) [GDM03] is a Java-based software engineering approach to build intelligent distributed agent systems. From a user perspective, the DECAF framework differs from other agent development frameworks by hiding the repeating complexities that agents systems rely on, such as communication structures, and letting users focus on the domain specific issues.

In DECAF, agents are defined in terms of their objectives and the set of actions that should be followed to reach each objective. DECAF supports both classical AI black and white objectives and weight oriented objectives [GDM03]. In the current DECAF implementation, objectives are defined in the task reduction trees (HTN [EHN94]).

In addition, DECAF comes with a graphical user interface called PlanEditor which facilitates design of agent objectives. The output of the graphical user interface is a plan file that is the programming language representation of the agent's objectives and actions. The plan file is an input to the DECAF core modules, which will be discussed later. Figure 5 shows an example agent plan designed in the PlanEditor.

The plan in Figure 5 contains one objective `updateDatabase` which has three input parameters: `DataOntology`, `SourceOntology`, and `URI`. The objective is reachable by completing three actions: `ExtractExternalInformation`, `TransformTheData`, and `LoadItIntoDatabase`. The outgoing message of each action is redirected to the input of the next action, thus forcing a sequence, except the last action's output which is connected to the output of the task and indicates that the end of the last action is the end of the task. In addition, the lines from the objective inputs to the `ExtractExternalInformation` action indicates those parameters should be delivered

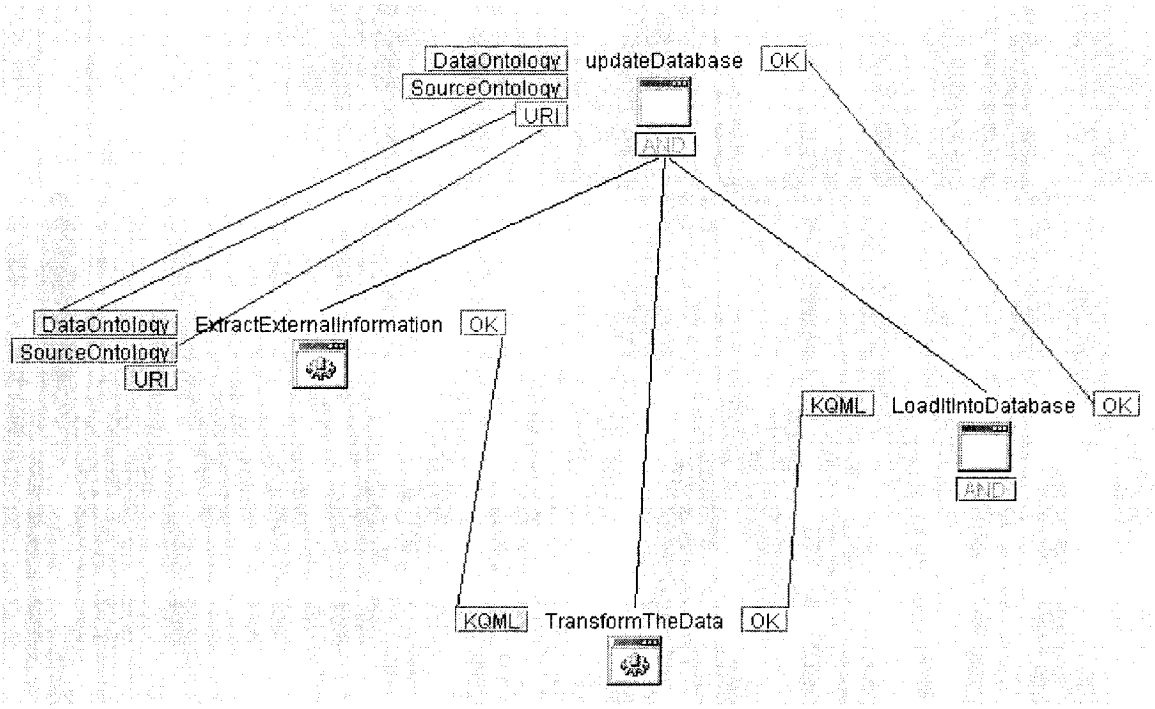


Figure 5: Sample DECAF agent plan in the plan editor

to the action.

As one can see in Figure 5, the agent task reduction trees are defined by putting together the plan building blocks. The building blocks are fairly standard and the current implementation of DECAF supports the followings:

- Task: Tasks are agent objectives, and they are the actions we can expect an agent to do. Tasks break down into subtasks and actions. The success of an agent's task depends on its Characteristic Accumulation Function (CAF). CAF indicates what sequence of successful subtasks or actions is good enough for a successful finish. Current implementation of DECAF supports the following CAFs:
  - AND: all the subtasks and actions should successfully finish.
  - OR: at least one subtask or action should successfully finish.
  - XOR: at most one subtask or action should successfully finish.

- SUM: the subtasks that maximize the task value should successfully finish.

- Action: Actions are implementation by a task. A task can have many actions, and the lines between the actions set the order of the actions.

- non-local-action: A non-local-action is something that the agent does not do but is necessary for the completion of its task. Non-local-actions can be tasks of other agents. What happens is that usually an outgoing message will be sent to the non-local-actions and the results are returned back as another message.

- Library: Libraries are reserved for non agent codes, and they are not currently implemented. Next releases of DECAF will fully support this type.

Defining agent plans is fairly straightforward, and it is out of the scope of this thesis. However, more details can be found in [McG01].

2.4.1 DECAF Architecture

The DECAF architecture comprises 5 modules and 7 data queues, as shown in Figure 6. Each of the modules performs a specific task within the DECAF system, and the modules work concurrently to execute agent plan files and response to the incoming KQML messages.

Agent Initialization

The agent initialization module is the starting point of an agent life cycle within the DECAF system. That is, the agent initialization reads the agent's plan file and extracts the agent's definition. Then it loads the agent's task reduction tree into the Task Templates Hashtable for further access.

In addition to loading the agent, the agent initialization calls the agent's `_startup` task to enable the agent to initialize itself. The last thing for the agent initialization

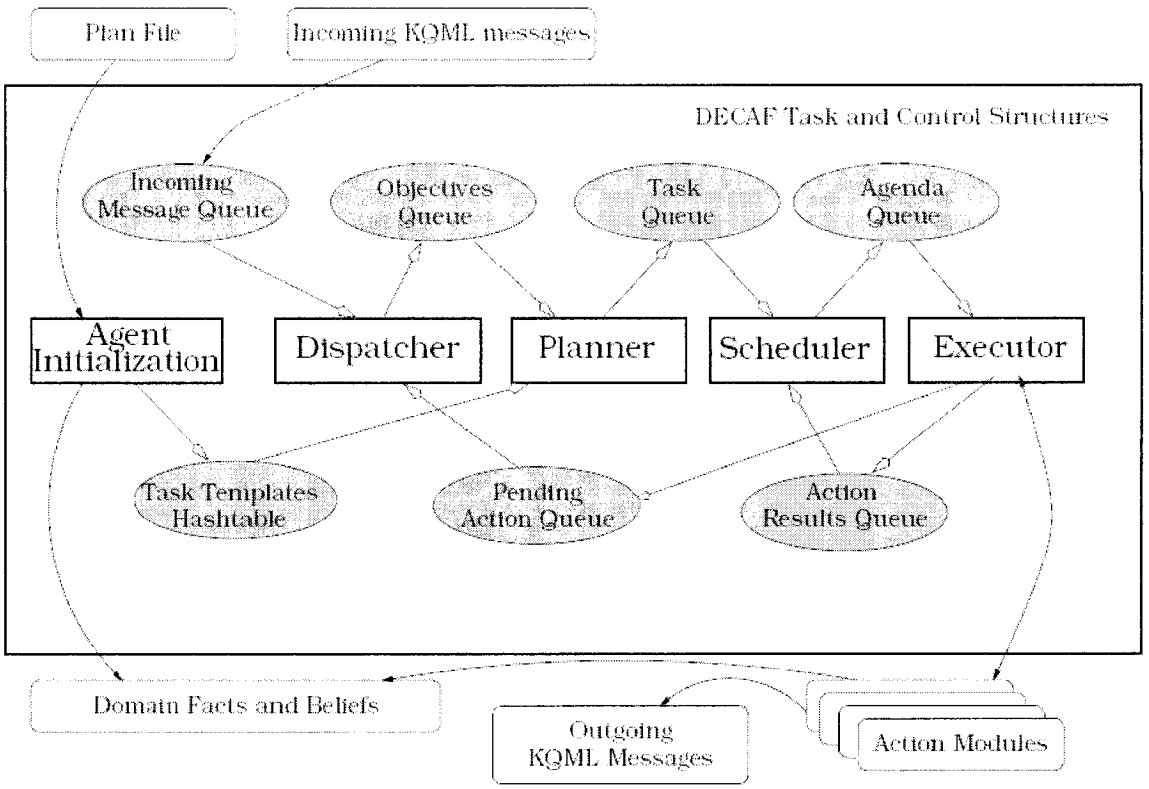


Figure 6: DECAF system architecture [GDM03]

is to register the agent in an Agent Name Server and establish the required network sockets and the communication for the agent.

Dispatcher

In DECAF, inter-agent communications are handled with KQML messages. Once a message created, it is put in the Incoming Message Queue. The dispatcher module is responsible for picking up the messages from the Incoming Message Queue and taking the right action. The dispatcher's current implementation makes three decisions based on the status of the message:

- If the message is a part of an ongoing conversation, the dispatcher finds the relevant pending action from the Pending Action Queue and enables the action by sending the appropriate message.

- If the message is a part of a new conversation, then the dispatcher creates a new objective and puts it in the Objective Queue so that the planner picks it up later.

- If the message contains any sort of error, the dispatcher generates appropriate error message and returns it.

Planner

The planner reads the Objective Queue and instantiates the appropriate task object for each objective. The planner uses the Task Templates Hashtable to find out which task should be instantiated. After instantiating the task, the planner puts the task in the Task Queue. The Task Queue contains a list of tasks to be completed.

Scheduler

The scheduler monitors the Task Queue to find out which tasks are ready to be executed and in what order they should be executed. In the current implementation the scheduler employs the first-come-first-served method to pick tasks, but an algorithm that considers the task cost function, duration, and some other utility functions is a suitable enhancement.

However, having selected a task from the Task Queue, the scheduler puts the task in the Agenda Queue so that the task will be executed.

Executor

The executor picks actions from the Agenda Queue and executes them. Once an action finishes, it returns an action outcome which will be placed in the Action Result Queue. After that, the framework distributes the results from the Action Result Queue to the tasks that are waiting for the results in the Task Queue, and the executor moves on processing next actions in the Agenda Queue.

2.4.2 Middle Agents

Middle agents are the agents that support other agents in an agent system. The middle agents facilitate agent activities that are common in agent systems. The DECAF framework has the following middle agents:

- Matchmaker: The matchmaker is an agent that lets other agents find each other. In essence, the matchmaker works as a yellow pages for agents.
- Broker: The broker agent acts as a white pages for agent services.
- Proxy: A proxy agent connects a local Java applet to the agents in a DECAF system.
- Agent Name Server: The agent name server is the agent version of the Domain Name Server, by which an agent name is resolved to an IP address and a port number. This agent is an integral part of DECAF's communication system.
- Agent Management Agent: The agent management agent allows listing of the agents in a distributed space. This will help administrators check agents' status while they are spread geographically. The agents need to be registered in the same Agent Name Server so that the Agent Management agent lists them.

2.4.3 Messaging in DECAF

The DECAF framework significantly relies on the agent communication. That is, agents activate each other's services by sending a message, and it is possible that an agent's objective is reached by making a chain of communication with other agents. Therefore, messaging plays an important role within the DECAF system, and the DECAF needs a messaging protocol that supports agent-to-agent talks and at the same time it supports continuous conversations.

DECAF uses a special formatted message called KQML. Each KQML message contains the following fields:

- Performative: this field specifies the type of the message. The performative is "achieve" unless it is one of the standard DECAF message types. The "achieve" performative is the way that an agent uses another agent's objective to reach its objective.
- Sender: this field indicates the name of the sender agent.
- Receiver: this field indicates the name of the receiver agent.
- reply-with: this field enables the support of continuous conversations. With this field, agents can talk to each other and refer to a conversation name so that they keep track of the conversation. This field is filled with the sender so that the receiver replies with the same value in the in-reply-to field.
- in-reply-to: this field reminds the receiver that this message is a part of an ongoing conversation. The sender puts the value of reply-with field in this field.
- language: this field is the messaging language. It is usually "DECAF".
- ontology: this field specifies the name space of the conversation. It is possible that agents use different ontologies; therefore, there is a need to agree on the content of the message. Agents that talk in the same ontology agree that they are using the same language and they have the same view of concepts. After all, this field ensures that the conversation goes on meaningfully.
- content: this field contains the content of the message. In the case that the performative of a message is "achieve", this field specifies the name of the task that should be done with the set parameters that should be passed to the task. The parameters are separated in the content by pairs of `:keyword value`, and

the standard parameter `:task task_name` specifies the name of the task in the receiver agent. Figure 8 shows an example of sending a KQML message in DECAF.

2.5 The Matchmaking Agent System

Matchmaking is the part of the DECAF framework that is responsible for connecting a service request to its appropriate server agent that fulfills the request. Therefore, the matchmaker component plays an important role within the agent system, and the better the matchmaker works the more efficient and useful the agent system is. However, the success of the matchmaker component is dependent on the success of the matching algorithm it employs.

Standard implementation of the DECAF matchmaker component uses basic string comparison techniques to find a match. That is, each service agent registers in the matchmaker component with a static string that describes the service, and the matchmaker adds the given service description in a local database. Then after, the matchmaker matches service requests with the string descriptions it has in the local database to find a match. Although this approach works, it is highly dependent to the clarity of the service descriptions and the terminology agreements between the seeker and provider agents. Al-Shaban et al. [ASH05] discuss that string comparisons are inefficient and incomplete, and they offer a replacement that is based on ontologies.

The Matchmaking Agent System [AS05] replaces the standard matchmaking mechanism of the DECAF, and brings the power of ontologies into the matching process. The logic behind the new matchmaker is that a special upper ontology, namely the service ontology, is shared between the seekers and the providers, and it contains all the possible service domains that the seekers and the providers would need[1]. After that, the providers are able to describe their service in terms of the

[1]The service ontology needs to be updated if a new area of service is introduced. In theory, it is

concepts in the service ontology, and the seekers are able to look for a service by posing their request in terms of the concepts in the service ontology. This way the service ontology is the language that the seekers, the providers, and the matchmaker use. However, this changes the standard way of registering in the agent system and looking for an agent. In addition, it enforces certain design requirements on the agents.

2.5.1 Provider Agents

Provider agents are the ones that offer a service. Therefore, they need to register in the matchmaker so that other agents find them and use their service. The Matchmaker Agent System requires provider agents to extends a certain plan file. This plan file, as shown in Figure 7, contains the required tasks that the matchmaker needs to trigger.

As shown in the Figure 7, the provider agents need to implement standard DECAF `_startup` and `_shutdown` tasks. These tasks bound the provider agents life cycle:

- The `_startup` task is responsible for registering the provider agent in the agent system. At this stage, the provider agent sends an `advertise` message to the matchmaker component, and tells the matchmaker to where its service belongs in the service ontology's tree of concepts. For example, a provider agent specialized in finding protein sequence similarities might tell the matchmaker that its service is classified under the `http://a.com/ontology#BLAST` concept. This causes the matchmaker to store the concept name and the agent name in its internal database for later retrieval. Figure 8 shows an example code snippet of a provider agent's advertisement by which the provider agent tells the matchmaker that it provides the BLAST service.

- The `_shutdown` message unregisters the agent in the matchmaker agent.

---

the best to define the service ontology in a way that it covers all services in a hierarchy so that it does not need frequent updates.

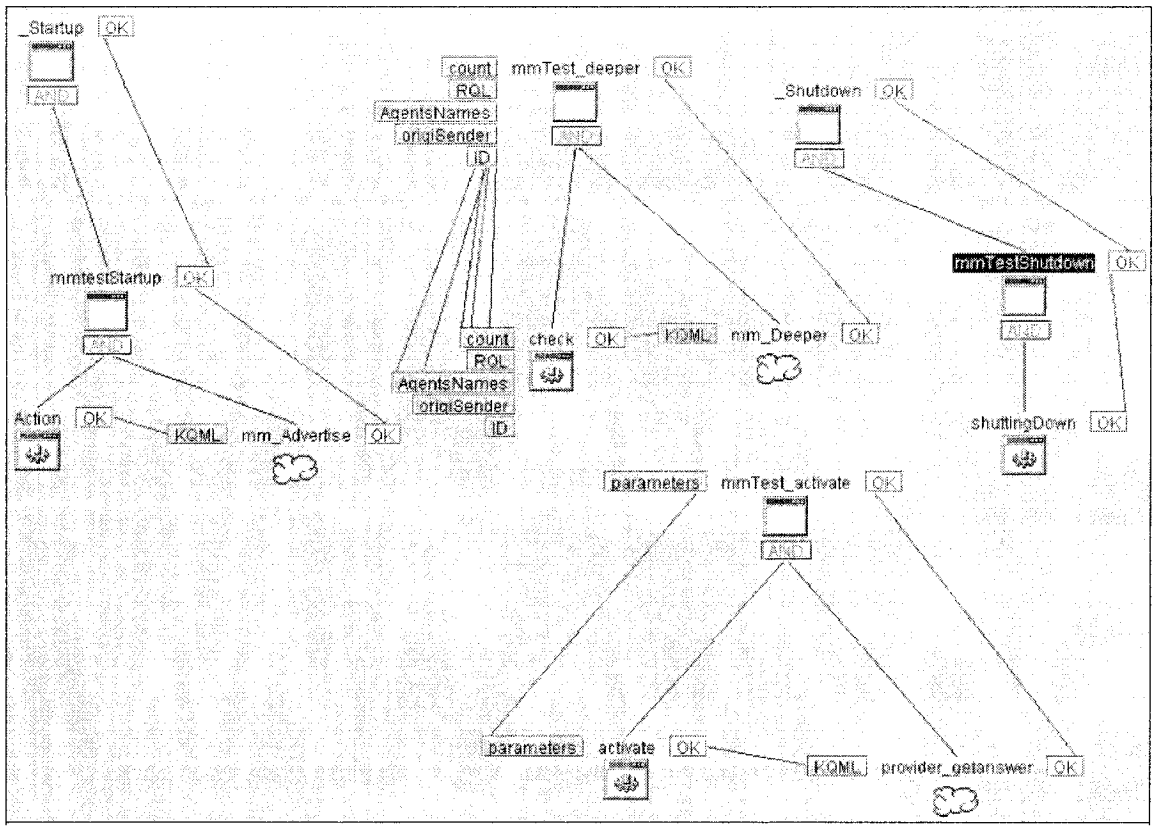


Figure 7: Provider agent's plan [AS05]

However, in addition to the standard tasks, the provider agents need to implement two additional tasks, `deeper` and `activate`.

The `deeper` is called by the matchmaker agent to check whether not the agent is able to fulfill the seeker's need. The importance of this task will be discussed in the next section where the role of the seeker agents is illustrated. The provider agent receives an nRQL query for this task, and it checks if its service meets the requirement specified in the nRQL query. The nRQL query may specify a data type or it may be a more specific description of the service the seeker is looking for. Note that the service ontology contains the general service domains, and it is not specific. As a result, the agents register themselves under general concepts in the service ontology, such as `http://a.com/ontology#BLAST`, and they can later in this task specify their specialization, if there is any. For example, a provider agent might run the given

```
public ProvisionCell Action(LinkedListQ Plist, Agent Local)
{
  String message = new String(Util.getValue(Plist, "MESSAGE"));

  KQMLmsg outKQML = new KQMLmsg(message);
  outKQML.addFieldValuePair("performative", "achieve");
  outKQML.addFieldValuePair("sender", Local.getName());
  outKQML.addFieldValuePair("receiver", "Matchmaker");
  outKQML.addFieldValuePair("ontology", "Matchmaker");
  outKQML.addFieldValuePair("language", "DECAF");
  outKQML.addFieldValuePair("content",
          ":task advertise"+
          " :keywords "+encode("http://a.com/ontology#BLAST")+
          " :ontology mmTest");

  return new ProvisionCell(outKQML.getKQMLString(),"OK");
}
```

Figure 8: An example of a provider agent's advertisement

nRQL query against a data ontology which defines the problem domain's elements and decide whether not it is able to fulfill the request. In the previous example, the provider agent might need to agree with the seeker agent that it finds the sequence similarity only between proteins and not genes.

In addition to the `deeper` task, the provider agents have to implement the `activate` task, in which the actual service is implemented. The seeker agents call `activate` task when they know to which agents they should talk.

More details of the provider agents can be found in [AS05].

2.5.2 Seeker Agents

The seeker agents are the computer softwares that need the provider agents service. The seeker agents can be a part of a bigger application, the implementation behind a web service, or the code behind a user interface. Likewise the provider agents, the

seeker agents have to extends a certain plan file as shown in the Figure 9.

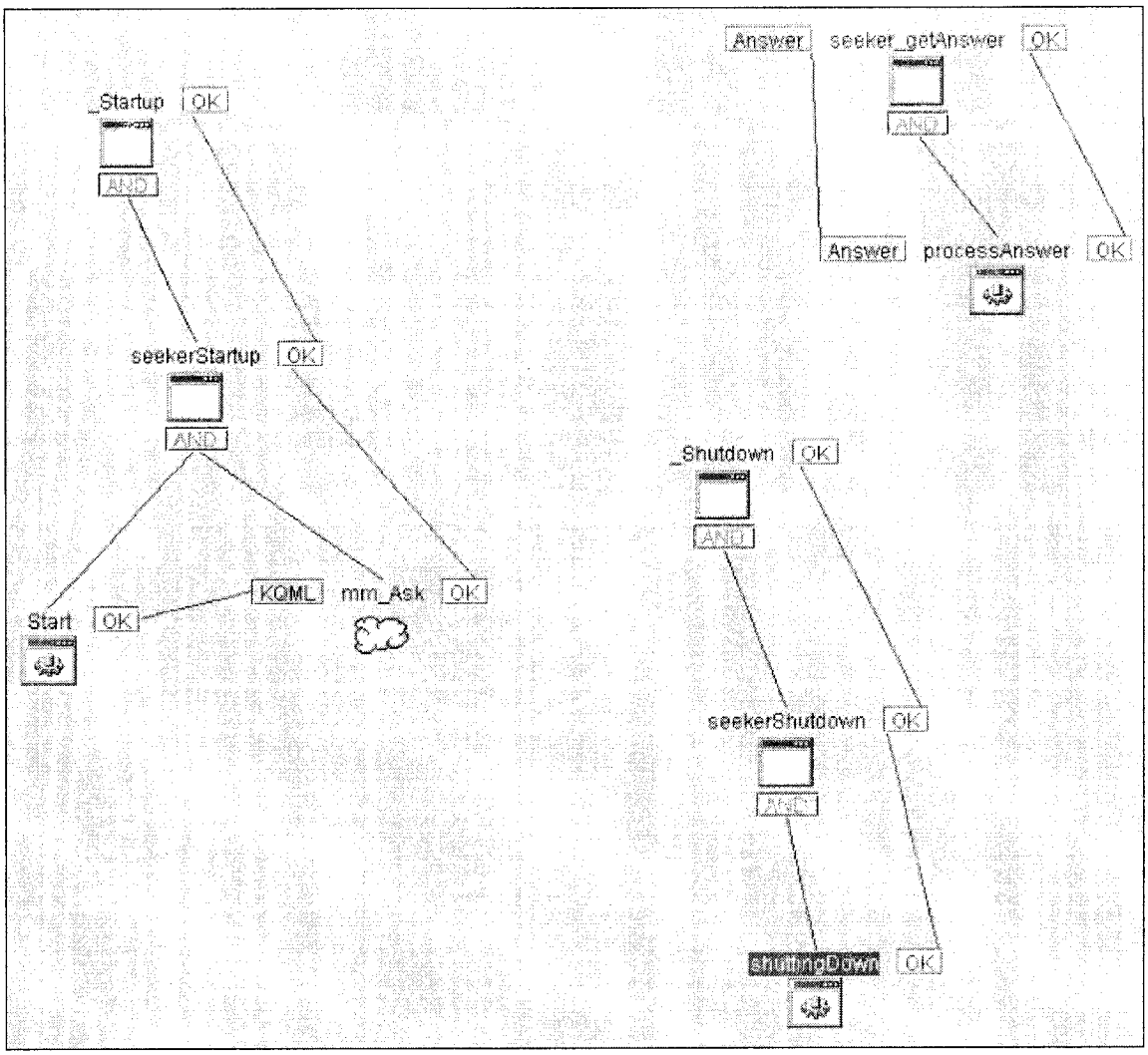


Figure 9: Seeker agent's plan [AS05]

The seeker agent's scenario starts by the standard `_startup` task, in which the seeker tells the matchmaker what service it needs. This is done by sending two queries to the matchmaker. The first query is a ConceptBasedQuery and the second query is an nRQL query. The ConceptBasedQuery queries the service ontology and it specifies the domain of the service. For example, the ConceptBasedQuery might be "I am interested in the agents that find sequence similarities". The RACER [HM01] query of such question looks like the one presented in the Figure 10.

```
(CONCEPT-DESCENDANTS |http://a.com/ontology#Sequence_Similarity|)
```

Figure 10: An example of a ConceptBasedQuery query

The nRQL query specifies the service in more details. Unlike the ConceptBasedQuery that is processed by the matchmaker, the nRQL query is processed by the provider agent. Therefore, there should be an agreement between the provider agent and the seeker agent so that the provider agent understands what the seeker agent is asking about. For example, the seeker agent might specify "I am interested in services that performs on the proteins". The nRQL query of such question looks like the Figure 11.

```
(RETRIEVE (?X) (?X |http://a.com/ontology#Protein|
                   |http://a.com/ontology#performs_on|))
```

Figure 11: An example of an nRQL query

The matchmaker agent first processes the ConceptBasedQuery by running the query against the service ontology. The result of the query is the list of the service concepts in which the seeker agent is interested. For example, the query in the Figure 10 returns the `http://a.com/ontology#BLAST` concept (assuming there is only this concept under the `http://a.com/ontology#Sequence_Similarity` concept). Then, the matchmaker queries its internal database to find out which provider agents have registered to provide this concept's service. Next, the matchmaker calls the `deeper` task of each provider agent and sends the given nRQL query to the provider agent. After that, the provider agent runs the nRQL query on an arbitrary ontology and retrieves the list of the ontology instances in which the seeker is interested. The provider agent examines the list of the ontology instances to see whether not it performs its service on that ontology instance. For example, the nRQL query

in the Figure 11 returns the `http://a.com/ontology#Protein` instance (assuming the arbitrary ontology is designed in such a way), and the provider agent returns positive answer to the matchmaker, confirming that it is able to provide the service for this type of entity[2]. After the matchmaker receives an answer from a provider agents it moves on to the next provider agent in the list till it finishes the whole list and extract a list of agent names that fulfill the seeker's need.

Once the matchmaker has the list of proper provider agents, it calls the `getAnswer` task of the seeker agent to let it know to which agent it should send `activate` message. At this point the matchmaking is done and the seeker agent is able to talk to the proper provider agents.

More details of the seeker agents can be found in [AS05].

2.6 BioXRT

Large volume of biological data is being produced each day, and the appropriate computer support to electronically publish and manipulate biological data is still missing. Diversity of available data formats and limited computer knowledge of biologists are some of the obstacles in the way of building such computer systems. BioXRT [ZDKS], previously known as XRT, is a simple but still powerful data warehouse that aims to fill this empty gap.

In a nutshell, BioXRT is a data warehouse designed to accept biological data and provide basic means of accessing it. Moreover, BioXRT is easy for biologists to use because it does not require advance computer/database skills, and it is proper for biological data because its input format covers most popular biological data formats.

Setting up BioXRT is pretty straightforward and it can be handled easily by going through installation steps of the installation guide. Even if the installation steps are

[2]The rest of this thesis shows how the provider agents use the FungalWeb ontology [SNBHB05] to provide biological services.

too much for biologists, they can be done once by a computer technician, and it is ready to load the input data. Loading data into BioXRT's database is also as easy as preparing the input files and running the appropriate BioXRT script to bulk load the data.

Loading Data

BioXRT accepts quite a few number of well-known formats such as Microsoft Excel, XML, and flat tab-delimited BioXRT. These formats mostly represent data in a tabular manner[3] which is a popular format among biologists [AVB01], and it is close to the way BioXRT represents the input data to the users. Furthermore, BioXRT provides a loader script for each of these input formats. These scripts are responsible for reading the input files and load them into BioXRT's internal database. The loader scripts erase the content of the internal database before loading the new set of data, so each load is a complete update.

`organisms.xrt`

ID	LongName	Order	Phylum	Kingdom
178477	Botryandromyces ornatus	Laboulbeniales	Ascomycota	Fungi
231773	Trichoderma sp. T-105	Hypocreales	Ascomycota	Fungi
205608	Buellia submuriformis	Lecanorales	Ascomycota	Fungi
193039	Patescospora separans	Jahnulales	Ascomycota	Fungi
4984	Bullera variabilis	Tremellales	Basidiomycota	Fungi
322976	fungal sp. 32.40	Unknown	Unknown	Fungi
307330	fungal sp. TRN236	Unknown	Unknown	Fungi
246499	Xylaria sp. F12	Xylariales	Ascomycota	Fungi
116810	Physcia albinea	Lecanorales	Ascomycota	Fungi

Figure 12: A sample BioXRT input file from FungalWeb project.

Each BioXRT input file stands for a class of data within the data warehouse. The class is the relevant BioXRT term for a table in the relational database's terminology.

[3]Even though XML is not a tabular format but it can be mapped to a tabular representation with trivial transformation rules.

Each input file declares the list of columns as attributes of the class, and each file should have an `ID` column, which is the unique identifier of entities within this class of data. Figure 12 shows an example of a BioXRT input file, in flat tab-delimited format, where columns are separated by the tab character, and the first row lists column names. The sample in Figure 12 shows a snippet of organisms class's input file in the FungalWeb data warehouse. The class represents the taxonomy of fungi species.

As described earlier, BioXRT avoids advance computer complexities so that biologists can make the most of it. Therefore, BioXRT does not support inter-data relations as perfect as most relational databases do. However, BioXRT supports the parent–child relation between classes. Such a relation in BioXRT is defined by using a special naming convention for the columns; The column that is keeping the parent id (foreign key) names as `P_ID/pc` where `pc` is the name of the parent class, and it keeps the values of the parent `ID` column. Figure 13 shows an example of this relation, where `gene.xrt` is representing child data of `org_feature.xrt`.

Accessing The Data

Accessing loaded data in BioXRT starts from the search screen, as shown in Figure 14. The search screen gives the users the ability to perform basic queries and keyword searches on a specific class of data. In addition, users can tweak their result set by selecting the appropriate columns. The search screen is simple enough to be understood by many users; however, it does not support constructing cross class queries.

The search screen needs to be configured. This level of configuration enables the publishers to partially hide their data and/or control how the search screen looks. Appendix A.1 shows the complete configuration file for FungalWeb's search screen.

Once a search is run within the search screen, the BioXRT renders the results using

org_feature.xrt

ID	TaxID	start	end	type	comment	source
1	162425	1213	1726	gene		MIT Broad Institute
2	162425	5806	3397	gene		MIT Broad Institute
3	162425	9062	6382	gene		MIT Broad Institute
4	162425	7822	1972	gene		MIT Broad Institute

gene.xrt

ID	P_ID/org_features	TaxID	GeneID	name	...
1	1	162425	AN0001.2	protein	...
2	2	162425	AN0002.2	protein	...
3	3	162425	AN0003.2	protein	...
4	4	162425	AN0004.2	protein	...

Figure 13: These two classes of data show an example of BioXRT's parent and child relation in the FungalWeb data warehouse. org_feature is where we keep features of an organism, one of which is a gene feature. Details of a gene feature is kept in gene.xrt which is a child class of org_feature.

its TBrowse engine, which is a simple tabular data renderer, as shown in Figure 15. It is possible to configure the TBrowse to link out the results to external sources, which is an essential feature for the biological data. This configuration should be done within the search screen's configuration file, where the columns for each class of data is defined.

In addition to the TBrowse result pages, BioXRT offers full page view of its data with the XView component. The XView is a XML-based data renderer. That is, XView reads a XML file as the template for the output, and it populates the template with the data it fetches from BioXRT's internal database and renders the output. XView is capable of joining multiple classes of data in the same page; therefore, it is a good way to show the parent and child data.

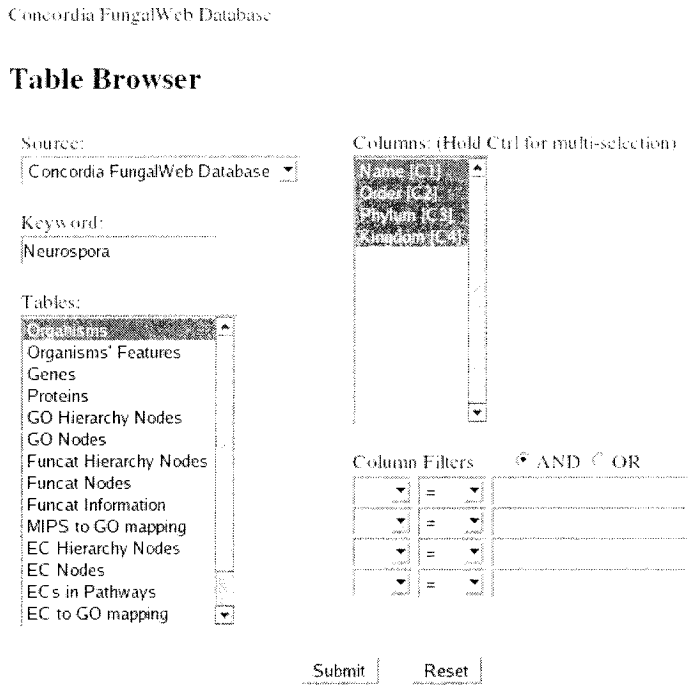


Figure 14: FungalWeb's BioXRT web query interface. The screen shot shows the list of available classes of data. Users can select a class and query the selected class against any arbitrary set of columns. They can also do a keyword search. The example shows how the search screen can be used to find all *Neurospora* species.

2.7 Extract Transform Load

A data warehouse is a collection of in-house and external data that have been gathered for the aim of providing a rich set of data (usually restricted to a specific domain), a high level of accuracy, and maximum data accessibility. A data warehouse can be any well-known database engine or it can be a hardware that is optimized for data access and large volume manipulation purposes.

One of the challenges in the process of building a data warehouse is populating the data inside the data warehouse and maintaining the data warehouse. Extract Transform Load (ETL) refers to the stage in data warehousing that data should be taken from an external source and integrated in the data warehouse.

The ETL, as the name explains, has three major steps:

- Extract: The first step is to obtain the data. Most data warehouse systems take

#	Name	Order	Phylum	Kingdom
1	Neurospora sublineolata	Sordariales	Ascomycota	Fungi
2	Neurospora calospora	Sordariales	Ascomycota	Fungi
3	Neurospora santi-florii	Sordariales	Ascomycota	Fungi
4	Neurospora hapsidophora	Sordariales	Ascomycota	Fungi
5	Neurospora sp. FGSC 8834	Sordariales	Ascomycota	Fungi

Figure 15: The example shows the result of the search for *Neurospora* species. The results are rendered by BioXRT's TBrowse engine which produces tabular data views in BioXRT.

the data from different sources. Therefore, the data comes in different formats and accessing them can be different. For example, a data might be available in a text file and the other might be the backup file of an RDBMS.

- Transform: It is possible that the data taken from outside does not comply with the data warehouse format or schema. In this case the data needs to be transformed to the current format and schema. The transformation rules can be fairly standard. Some of such rules are:
  - Loading only certain columns of the source data
  - Changing parameters in the case they are different from the ones in the data warehouse. For example, if a source data keeps A as "active" and D as "deleted" but the data warehouse keeps boolean values.
  - Parameterizing values in the case the source does not parameterize.
  - Computing new values based on the values from the source data. For example, retrieving GenBank IDs from NCBI for genes.
  - Merging data from different tables/sources.
  - Summarizing multiple rows of the source data.
  - assigning primary keys to the source data.

– Creating multiple rows of data for a single row from the source.

- Load: This step loads the transformed data into the data warehouse.

In recent years, the ETL has turned to a standard process of data warehousing applications, and there are working ETL libraries for most programming languages. CloverETL (`http://cloveretl.berlios.de/`) is an example of such libraries.

CloverETL is a Java based ETL library that enables Java applications to build an ETL engine by putting together ETL building blocks. In CloverETL, application designers need to form an ETL graph by connecting ETL components. The graph visualizes flow of data during the ETL process.

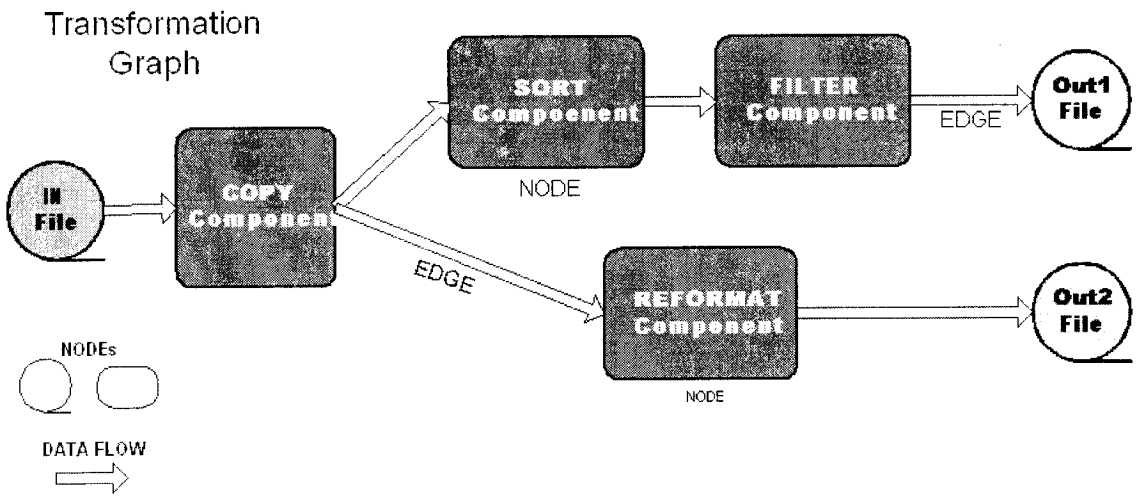


Figure 16: An example ETL graph taken from the CloverETL website.

The ETL graph has some input nodes that are CloverETL supported input formats, and it has some output nodes that can be either text files or any standard relational database. The ETL components connect the input nodes to the output nodes. Figure 16 shows how Copy, Sort, Filter, and Reformat components have been used to transform the input data to the output, and how the transformation has been split in two paths of the graph.

Chapter 3

The Data Warehouse

Biological databases and analysis tools are the result of research efforts, and they are diversely scattered through the world. Easy access to biological databases and analysis tools is an important factor in the success of new biological initiatives. However, the distributed nature of biological databases and analysis tools makes the process of knowledge extraction inaccurate and hard-to-achieve. That is, these databases provide different focuses of the same data, they employ different representation format (plain text, formatted text, XML, and so on), they are available in different forms (relational databases, text files, XML, and so on), and they have different approaches to data manipulation [SKSB00].

As a result, new enhancements in biology and bioinformatics are highly dependent on the successful development of integrated database systems. The ongoing work to integrate multiple databases and analysis tools categorizes integrated database systems into three distinguishable groups. Yona et al. [BY06b] explain that most integrated database system are either portal oriented systems, mediators, or data warehouses.

Portal oriented systems are characterized by the fact they do not store the original data, but they store indexes of the data in the external databases. Therefore, portal

oriented systems are fast in searching through their integrated data. In addition to the indexes, portal oriented systems keep a link of the original data so that they can navigate the user to the original data. In theory, users need to follow the link to access the original data. However, it is also possible that the link is transparent to the user, and the system follows the link to give the user the original data (it happens each time the user tries to access the data and the system does not offload the original data). The integration of cross-database relations is an optional characteristic of portal oriented systems. SRS [EA92] and Entrez [SEOK96] are examples of portal oriented systems.

Mediator oriented systems are like the portal oriented systems in the sense they do not store the original data of the external databases. However, mediator systems keep a schema of the external databases, and they talk to the external databases to run a query. What the mediator systems do is that they let the user query the integrated database as if it were a single database. Then the mediator system uses the schema to find out which external databases should be consulted to fulfill the user query, and it creates and sends the appropriate query to each source. Upon receipt of the result from each source, it uses the schema again to integrate the results and return it to the user. In theory, mediator systems easily integrate new databases. However, mediator oriented systems' speed is highly dependent to the speed of external sources, and they are not performance efficient when multiple sources are involved. DiscoveryLink [HKR+00], BioMediator [MHTH01], and TAMBIS [BBB+98] are examples of mediator oriented systems.

Warehouse oriented systems are an alternative to mediator oriented system and portal oriented system. Warehouse oriented systems integrate the data from the external database inside their internal database. Warehouse oriented systems introduce a new data schema which covers the data schema from all external databases and supports the entity relation between data of the external databases. Warehouse oriented

systems are performance effective, and they are significantly more efficient in complex queries than the other two systems. However, maintenance is a major concern for warehouse oriented systems. Biozon [BY06b] and GUS [DCB+01] are examples of warehouse oriented systems.

As a part of the FungalWeb project this research aims to build a data warehouse system that provides access to fungal genomics data as instances of the FungalWeb ontology. Moreover, the data warehouse system includes a data warehouse and an agent system which manipulates the data warehouse and provides access to it. In this chapter we focus on the data warehouse and the next chapter is dedicated to the agent system.

3.1 Data

The data warehouse holds data of a selected range of fungal species that have either industrial significance or research importance. Table 3 shows the list of currently integrated fungal species.

In addition to the fungal species, the data warehouse includes some classification information which is independent of fungal species, but it elaborates the features and functions of the selected fungal species. In a nutshell, the data warehouse contains the taxonomy of fungal species, fungal genes and proteins, Gene Ontology classification, FunCat, Enzyme Classification, metabolic pathways, InterPro, InterProScan, and Saccharomyces Genome Database data. Appendix B explains the structure of the data file for each of these data.

Species	NCBI Taxonomy ID
Aspergillus niger	5061
Aspergillus nidulellus	162425
Laccaria bicolor	29883
Coprinus cinereus	5346
Trichoderma reesei	51453
Pichia stipitis	4924
Phanerochaete chrysosporium	5306
Nectria haematococca	140110
Neurospora crassa	5141
Magnaporthe grisea	148305
Botrytis cinerea	40559
Ustilago maydis	5270
Aspergillus terreus	33178
Rhizopus oryzae	64495
Coccidioides immitis	5501
Saccharomyces cerevisiae	4932
Candida albicans	5476
Cryptococcus neoformans	40410
Eremothecium gossypii	33169
Gibberella zeae	5518
Schizosaccharomyces pombe	4896

Table 3: Fungal species in the data warehouse

3.1.1 Taxonomy

A taxonomy classifies species in terms of their order, phylum, and kingdom in a hierarchical tree. In a taxonomy, parent species generalize children species. For example, *Fungi* generalizes *Ascomycota, Basidiomycota, Chytridiomycota, Fungi incertae sedis, Glomeromycota, Microsporidia, Zygomycota,* and *Unclassified Fungi.* In the data warehouse we keep the taxonomy of fungal species, and the taxonomy has been taken from NCBI's taxanomy database (`http://www.ncbi.nlm.nih.gov/Taxonomy/taxonomyhome.html/`).

3.1.2 Genes and Proteins

Genes and proteins are the fundamental information of a species. In an organism, genes carry the information of the organism, and they control the physics and behavior of it. Gene products are of significant importance in study of an organism, and they explain the source of organism changes. Proteins are one of the genes' products. Proteins are the chemically active units of organisms, and they cause functions of organisms.

The data warehouse includes the genes and the proteins of *Neurospora crassa, Coprinus cinereus, Aspergillus nidulellus, Magnaporthe grisea, Botrytis cinerea, Ustilago maydis, Aspergillus terreus, Rhizopus oryzae, Saccharomyces cerevisiae, and Coccidioides immitis* fungal species. In addition, the data warehouse includes the proteins of *Aspergillus niger, Laccaria bicolor, Trichoderma reesei, Pichia stipitis, Phanerochaete chrysosporium, and Nectria haematococca* fungal species, for which the genes were not available. As a step toward linking the genes and proteins in the data warehouse to the external sequence databases we have used NCBI BLAST service [MM04] to find the GenBanks identifiers of the proteins and genes and store them in the data warehouse. GenBank identifiers are widely accepted unique identifier of sequences.

The genes and the proteins have been taken from multiple sources, some which are Fungi Genome Initiative at MIT Broad Institute (`http://www.broad.mit.edu/annotation/fgi/`) and the US Department of Energy Joint Genome Institute (`http://www.jgi.doe.gov/`).

3.1.3 Gene Ontology

Gene Ontology (GO) [ABB+00] is an ongoing community work to standardize the vocabulary of genes and gene products. Like other classification schemes, GO categorizes the gene vocabulary in a tree which starts with three major categories: *biological*

process, *cellular component*, and *molecular function*. The GO schema has been originally started by collecting vocabulary from different databases such as Saccharomyces Genome Database (SGD) [HBC+] and Mouse Genome Database (MGD) [EBK+05]; however, it is the primary classification for the new biological databases. In addition to the classification, GO offers the mapping between GO classification and other classifications such as EC and FunCat. The mapping helps with relating the other classifications to each other through the GO classification.

The data warehouse stores the GO classification and the mappings from the GO classification to EC, InterPro, and FunCat classfications. Where possible, the GO has also been used across the stored data.

3.1.4 Gene Ontology Annotation

Genes and their products can be annotated according to the Gene Ontology. The annotation involves the comparison of a source gene or a gene product to other genes or gene products. This helps characterizing unknown genes and unknown gene products by relating them to known ones. Gene Ontology annotation is a standard process, for which more information is available online at `http://www.geneontology.org/GO.annotation.shtml`.

The data warehouse includes the result of Gene Ontology annotation of *Aspergillus nidulans*, *Aspergillus niger*, *Candida albicans*, *Cryptococcus neoformans*, *Eremothecium gossypii*, *Gibberella zeae*, *Magnaporthe grisea*, *Neurospora crassa*, *Saccharomyces cerevisiae*, *Schizosaccharomyces pombe*, *Ustilago maydis*, and *Mycotorula lipolytica*.

3.1.5 FunCat

The Functional Catalogue (FunCat) [RZM+04] is a quite recent effort from Munich Information Center for Protein Sequences (MIPS) to standardize protein function vocabulary. That is, FunCat categorizes functions of proteins from *prokaryotes, unicellular eukaryotes, plants, and animals* into a tree. Existence of FunCat helps biologists and bioinformaticians assign function codes to regions of proteins, and exchange their assignments without the need to clarify the functions.

The data warehouse in this research stores the FunCat classification, and where applicable it stores the mapping between FunCat definitions to the Gene Ontology definitions. This way, other classifications can be mapped to the available FunCat classification through the Gene Ontology.

In addition to the classification, the data warehouse stores the assigned FunCat definitions to the fungal proteins. These assignments help researchers find the function of unknown proteins by finding the similarity between the proteins with unknow functions and proteins with known function.

3.1.6 Enzyme Classification

Enzyme classification [Bai00] is yet another hierarchical classification for enzymes. The classification is based on the recommendations of Nomenclature Committee of the International Union of Biochemistry and Molecular Biology (IUBMB, `http://www.iubmb.unibe.ch/`). The enzyme classification assigns a unique EC (Enzyme Commission) number to each enzyme type so that it can be used and referred to in other databases. Other fields include: name, alternative names, catalytic activity, cofactors, link to protein sequences in the SWISS-PROT, link to human diseases associated to an enzyme's deficiency.

The data warehouse stores the EC classification, and it uses EC numbers where applicable. In addition the data warehouse includes the EC to GO mapping so that

it can be mapped to other classifications through GO classification.

3.1.7 Metabolic Pathways

A metabolic pathway is a chain of chemical reactions that are catalyzed by enzymes. Metabolic pathways may result in a final chemical product, or they may trigger other metabolic pathways. Study of metabolic pathways has an important significance in the understanding of what happens in a living organism.

The data warehouse integrates the metabolic pathways of the KEGG database [KEG99]. Furthermore, the main focus of the data warehouse in metabolic pathways is which enzyme types are used in the metabolic pathways.

3.1.8 InterPro

InterPro [AAB+01] is an integrated source of signature databases such as PROSITE [HBB+06], PRINTS [Att02], ProDom [CSGK00], and Pfam [BBD+00]. Signature databases help with understanding newly annotated proteins by examining their similarities with characterized proteins. The InterPro database covers the information of protein families, protein domains, and functional sites. Furthermore, the Interpro database contains the the functional description, annotation, literature references, and the link to the integrated databases for each entry. The InterPro also comes with a classification schema which is a tool for sharing its vocabularies, as it is in the case of other classifications. Furthermore, the terms in the InterPro classification are mapped to the terms in the GO classification.

The data warehouse contains the InterPro classification, and its mapping to the GO classification. We use the InterProScan (see next section for more information) to predict functions of fungal proteins and genes against the InterPro database.

3.1.9 InterProScan

InterProScan [PSK+05] is an analysis application which predicts the functions of an unknown protein or gene by examining its similarity against the data in the InterPro database. InterProScan in essence is a wrapper around the following applicaitons:

- FingerPRINTScan [SFA99] for searching PRINTS database.
- ProfileScanner for searching the profiles of the PROSITE database.
- Ppsearch [Fuc94] for pattern matching in PROSITE database.
- HMMPfam for scanning protein sequences in Pfam database.

InterProScan is able to perform arbitrary set of the above applications on a sequence and report any similarities while including the evidence for each similarity.

The data warehouse contains in-house computed results of the InterProScan application on all the fungal sequences it has. The InterProScan has been run by all its application in order to build most comprehensive set of fungi computed signatures.

3.1.10 Saccharomyces Genome Database

Saccharomyces Genome Database (SGD) is a comprehensive source of Saccharomyces cerevisiae yeast. The SGD database includes genes, proteins, homologies, functions, and expression data and it is maintained and curated by the SGD people. The SGD database is freely available at `http://www.yeastgenome.org/` web site, and it is available in simple text format.

Table 4 shows the list of the data we have integrated from the SGD database. The data has been transformed from the SGD flat text files to the BioXRT's format.

Data Class	Description
annotation_change	It specifies the features that have been either removed or merged into another feature.
best_hits	Blast results for the genes and proteins
biochemical_pathways	Biochemical pathways
chromosome	Chromosomes and their sequence
chromosome_length	Length of the chromosomes
clone	Contains information about yeast clones from Washington University in St. Louis and the ATCC.
dbxref	The relation between SGD entries and other database such as SWISS-Prot
domains	List of predicted domains
emotif	List of motifs
gene_association	GO annotations
gene_literature	The literature reference to the yeast genes
genetic_map	Genetic mapping data
go_protein_complex_slim	It maps yeast gene products to the Macromolecular Complex GO-Slim terms.
go_slim_mapping	It maps yeast gene products to the GO-Slim terms.
interations	
intergenic_seq	List of sequences that are not associated to a feature
orf_dna	ORF DNAs
orf_geneontology	GO annotations for ORFs
orf_protein	
orf_sequence	
pdb_homologs	Homologies with the PDB data
pORF_Yeast_GP	List of primers for the partial ORFs
protein_properties	
psi_blast	Results of the PSI Blast
registry_genenames	A central repository for the Saccharomyces gene names
SGD_features	List of chromosomal feature in the SGD
yeast_est_seq	List of EST sequences
yeast_gb_seq and yeast_nrpep_seq	Saccharomyces sequences from other databases
yeast_GP_seq	List of primers for the entire ORFs

Table 4: List of the yeast data stored in the data warehouse

3.2 Architecture

The architecture of the data warehouse is fairly simple and straightforward. We use BioXRT as our data warehouse backbone. BioXRT is chosen because it has a simple structure, it is easy to update its content, it is popular within biologists' community, and it provides a simple but yet comprehensive web interface to access the data warehouse.

The access to the data warehouse is limited to the agent system boundary. Agents and the BioXRT web interface are the standard ways of accessing the data. In the long run, agent are supposed to also provide web service access to the data. In terms of maintenance, again agents are responsible for maintaining the data inside the data warehouse. Therefore, the data storage and access is totally transparent to the users.

The data warehouse is populated and updated by the agent system. Each update of the data warehouse takes bulk loading of the whole data sets which takes about half an hour with the current datasets. Therefore, the preferred update method is nightly schedules as new data is provided.

3.3 Loading the Data Warehouse

In the early stages agents were not been used to populate the data warehouse, but a set of Java applications transformed the data from its original form to the BioXRT flat-tab-delimited text files. However, the same set of Java applications are preferred to be used in the ETL process of the data transformation and population.

Certain transformation steps were applied in order to transform each described set of data to the BioXRT format. Some of the trasnformation steps are as described below (the code for the transformers are available online at `http://www.cs.concordia.ca/~f_kohant/thesis/tsource.zip`:

- Genes and proteins: Many sequence databases are available in the FASTA format. Therefore, a separate FASTA converter for genes and proteins has been developed to transform sequences from the FASTA format to the BioXRT format. The FASTA converter reads the standard FASTA files with a set of sequences and assigns each sequence a unique identifier. The next step in transforming genes and proteins is to find the GenBank identifier for each sequence. A Java toolkit has been developed for this purpose (accessible at `http://www.cs.concordia.ca/~f_kohant/ncbiblast/`). The Java toolkit enables users to queue their BLAST jobs on the NCBI's BLAST server, and retrieve the results. The transformer code for this part uses the results of the NCBI's BLAST for finding the GenBank identifiers.

- InterProScan data: After the gene sequences and protein sequences are ready, they are submitted to the InterPro's InterProScan server to find the matching regions of the submitted sequences against the InterPro database members. The transformer for this part uses the EBI web services [PSK+05] to post InterProScan jobs on the server. Upon completion, the results of each job is returned as an XML which the transformer parses and stores in a separate BioXRT file.

- Metabolic Pathways: KEGG offers its dataset in XML format. An specialized KEGG XML transformer was developed for this part to transform the metabolic pathways to the BioXRT format.

- Classification data: Some classifications such as Gene Ontology are available in the XML format. A specialized XML transformer was developed for each of them. The rest were available in text format for which the appropriate transformers were developed. The mapping between the classifications and GO classification is also available in text format for which again the appropriate transformers were developed.

After the BioXRT files are ready, they are bulk loaded in the BioXRT by the provided shell scripts.

3.4 Facts

Table 5 lists the size of the 7 biggest data sets loaded into the data warehouse. The total size of the data warehouse is currently 3,828,228 records. However, the data is still being transformed and it is expected the total amount will come to around 10 million records.

The current amount of data takes around 6 hours to load into BioXRT. Loading time is the time BioXRT needs to parse the input and store the records inside its internal database. The load time is once per update, and the data is accessible instantaneously after it is loaded. Note BioXRT startup is fast and it is not effected by the amount of data it has in its storage. It is worthy to mention BioXRT spends most of the loading time indexing the input data and not parsing it. Therefore, a faster and tuned database engine will reduce the 6 hours loading time.

Data	Size (records)
Gene Ontology	975,637
Proteins	248,085
GO Annotations	243,729
Genes	109,006
FunCat Signatures	53,045
SGD	939,651
InterProScan (Proteins)	1,157,662
...	...
TOTAL	3,828,228

Table 5: Data warehouse size in terms of records

Finally, the data warehouse takes around 5 gigabytes of disk space in the BioXRT's internal database, MySQL.

3.5 Access

The data warehouse is currently accessible by the BioXRT web interface. As an example, imagine one is looking for "All Neurospora Crassa proteins whose function has something to do with Hydrolase". He/She will need to go to the query page, as shown in the Figure 17, and select the FunCat information data type from the list in the bottom left side of the query page.

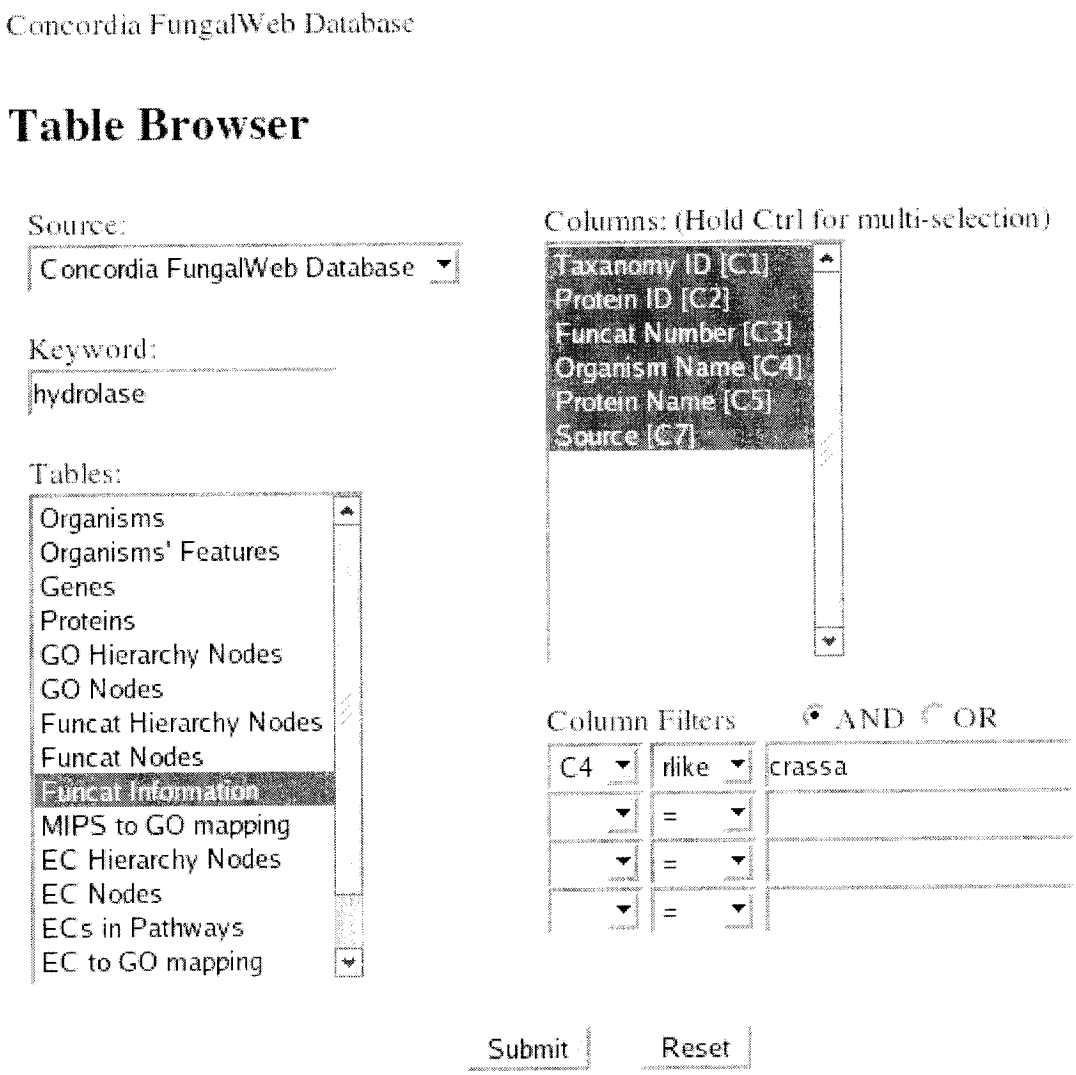


Figure 17: Example query in the BioXRT interface of the data warehouse

Thenafter, he/she will need to specify the query criteria. The bottom right side controls of the query page provide column based criteria. In this example, proteins

of *Neuropora Crassa* is the first part of the question, so he/she will need to set the forth column (organism name) to contain *Crassa* word. In addition, having a relation with *Hydrolase* function is the next part of the question. Lets assume that the user is not so sure in which column he/she should look for *Hydrolase*. As a result, he/she will need to put *Hydrolase* in the keyword text box at the top left side of the query page. This lets the user look for *Hydrolase* in all of the columns.

Pressing submit will take the user to the results shown in Figure 18, in which the proteins are listed. Note in this class of data, FunCat numbers are used to describe the actual function of the protein. One may also go further and look for the corresponding GO classification of the protein function.

#	Taxonomy ID	Protein ID	Funcat Number	Organism Name	Protein Name
1	5141	6nc350_510	01.01.09.05.02	N.crassa	probable fumarylacetoacetate hydrolase
2	5141	6nc350_510	01	N.crassa	probable fumarylacetoacetate hydrolase
3	5141	6nc350_510	01.01	N.crassa	probable fumarylacetoacetate hydrolase
4	5141	6nc350_510	01.01.09	N.crassa	probable fumarylacetoacetate hydrolase
5	5141	6nc350_510	01.01.09.05	N.crassa	probable fumarylacetoacetate hydrolase
6	5141	49d12_120	14.07.05	N.crassa	related to ubiquitin carboxyl-terminal hydrolase
7	5141	49d12_120	14	N.crassa	related to ubiquitin carboxyl-terminal hydrolase
8	5141	49d12_120	14.07	N.crassa	related to ubiquitin carboxyl-terminal hydrolase
9	5141	2nc560_060	01.05.01.01	N.crassa	probable epoxide hydrolase
10	5141	2nc560_060	01	N.crassa	probable epoxide hydrolase
11	5141	2nc560_060	01.05	N.crassa	probable epoxide hydrolase
12	5141	2nc560_060	01.05.01	N.crassa	probable epoxide hydrolase
13	5141	2nc850_060	01.06	N.crassa	acetyl-CoA hydrolase
14	5141	2nc850_060	70	N.crassa	acetyl-CoA hydrolase
15	5141	2nc850_060	70.03	N.crassa	acetyl-CoA hydrolase
16	5141	2nc850_060	01	N.crassa	acetyl-CoA hydrolase

Figure 18: The results of the example query

In addition to the BioXRT web interface of the data warehouse, one may use SQL to query the data inside the BioXRT's internal relation database.

Chapter 4

The Agent System

Agent systems have been around in the Life Sciences for a while, and the currently available agent systems such as BioMAS [DKS+02] have proven useful on manipulating dynamic databases [Bur]. As a part of the FungalWeb project, in this research an agent system has been introduced in order to maintain the data warehouse and provides access to the data in the data warehouse.

The agent system has been developed in DECAF agent framework. Therefore, it is a plan-driven agent system that benefits from intelligent scheduling and distributed nature of the DECAF.

In terms of agent types, the agent system is comprised of two different sets of agents. The first set of agents accesses and updates the data warehouse. This serves as instance data for the FungalWeb ontology [SNBHB05]. The entities in the data warehouse match the foundational concepts in the ontology. The agents that update the data warehouse have explicit knowledge of a range of data resources: the agents know how to query the data resource, transform results, and load the data into the data warehouse. The agents that access the data warehouse use the ontology query language, nRQL [HMW04], for RACER [HM01]. One can also access the data warehouse through the native interface.

The second set of agents that is being developed is to create a distributed data space where agents can access the entities (matching the concepts in the ontology) either via the data warehouse or via the distributed data resources. This is transparent to the users.

The agent system is unique in its kind due to its general agent types. Although the current implementation provides access to fungal genomics information, the agent system can be expanded to cover other information that is semantically categorized in an ontology. This compares to the agent types in other currently available agent systems. For example, the BioMAS agents are responsible for: basic sequence annotation, functional annotation, and processing of expressed sequence tags (ESTs) [Bur]. The effort for adding the support for a new service in such agent systems requires design change and code implementation while we argue that the effort required to integrate new service or new databases is competitively minimal in our agent system.

In terms of design, all of the agents in the agent system extend the provider plan of the matchmaker agent system (see section 2.5), and they register themselves in the matchmaker system using concepts from the service ontology. In addition, the agents refer to the FungalWeb ontology to understand the exact type of the service the seeker agents are inquiring about. Recall that the design of the matchmaker agent system encourages use of two ontologies for finding an agent service. The first ontology narrows down the list of candidate agents based on the domain of service they provide, and the second ontology is in the hands of the provider agents to understand the seeker agents' inquiry. After all, the second ontology is an agreement between the provider agents and the seeker agents on the specialization of general service domains in the service ontology.

Using two ontologies for finding agent services introduces two interesting issues that are worth discussing here. First, service seekers and service providers should have the same service ontology and the data ontology (a more appropriate term for

the second ontology in this research) so that they can work together, otherwise there need to be a matching mechanism to match concepts in one ontology to the other. However, this is not the case in the agent system we are designing because both sides are sharing the same service ontology for the matchmaking agent system, and for the data ontology which is the FungalWeb ontology here. As a result, it is always guaranteed that both sides are using the same set of concepts.

However, one may argue that convincing people to use the same set of ontologies is an optimistic assumption. We contend that biological ontologies are now becoming results of community work, and they are maintained to be consistent in the future. Open Biomedical Ontologies (OBO) [OBO] is an example of such community works. As a result, individual groups that are building their own ontology can extend OBO's ontology and remain consistent with other ontologies in the domain, so the FungalWeb ontology is going to be.

The second interesting issue is how the service ontology and the data ontology can be employed together to locate a service. Currently what we do is that we define generic services in the service ontology, regardless of concepts in the FungalWeb ontology. After that, generic services have parameters by which service concepts are linked with data concepts in the FungalWeb ontology. For example, a `retrieve` service on the service ontology might have a class parameter, like gene, which comes from the FungalWeb ontology. Essentially this parameter specifies the service, and it is passed to the provider agent. The advantage of this approach is that we do not need to worry about ontology maintenance, and the drawback is that services are not specific, therefore harder to implement and maintain. However, there is another approach that merges the FungalWeb ontology into the service ontology, and the result is a third ontology that contains specialized versions of the generic services. For example, imagine that `retrieve` service is a generic service in the service ontology, and it operates on `Gene` and `Protein` concepts from the data ontology. After merging

with the FungalWeb ontology, the third ontology will have two versions of retrieve service: `retrieve_gene` and `retrieve_protein`. The benefit of this approach is that services are specific, but the drawback is that the third ontology should be maintained against the changes in either of the ontologies.

4.1 The Service Ontology

A part of this research is the integration of a biological service ontology into the matchmaker's service ontology. This enables our provider agents register in the matchmaker and seeker agents find them.

We extend the BioMoby [WL02] service ontology, as shown in Figure 19. This makes the service ontology consistent with the currently developed service ontologies in the field of biology. For the moment, the service hierarchy in Figure 19 covers all the services of the developed agents in the agent system. However, integration of new services may need modification of the service ontology.

4.2 The Architecture

The architecture of the agent system has been influenced by the matchmaker agent system's requirements. Figure 20 shows the architecture in respect to the provider agents and their accessibility.

The lines in the Figure 20 show the communication between the components, and the directions specify the direction of either message or data request. A standard scenario starts with the advertisement of a provider agent and registration of the provided service concept from the service ontology in the matchmaker's database. Upon completion of the registration, a seeker agent can send a message to the matchmaker providing two queries, a RACER query and an nRQL query. Having the

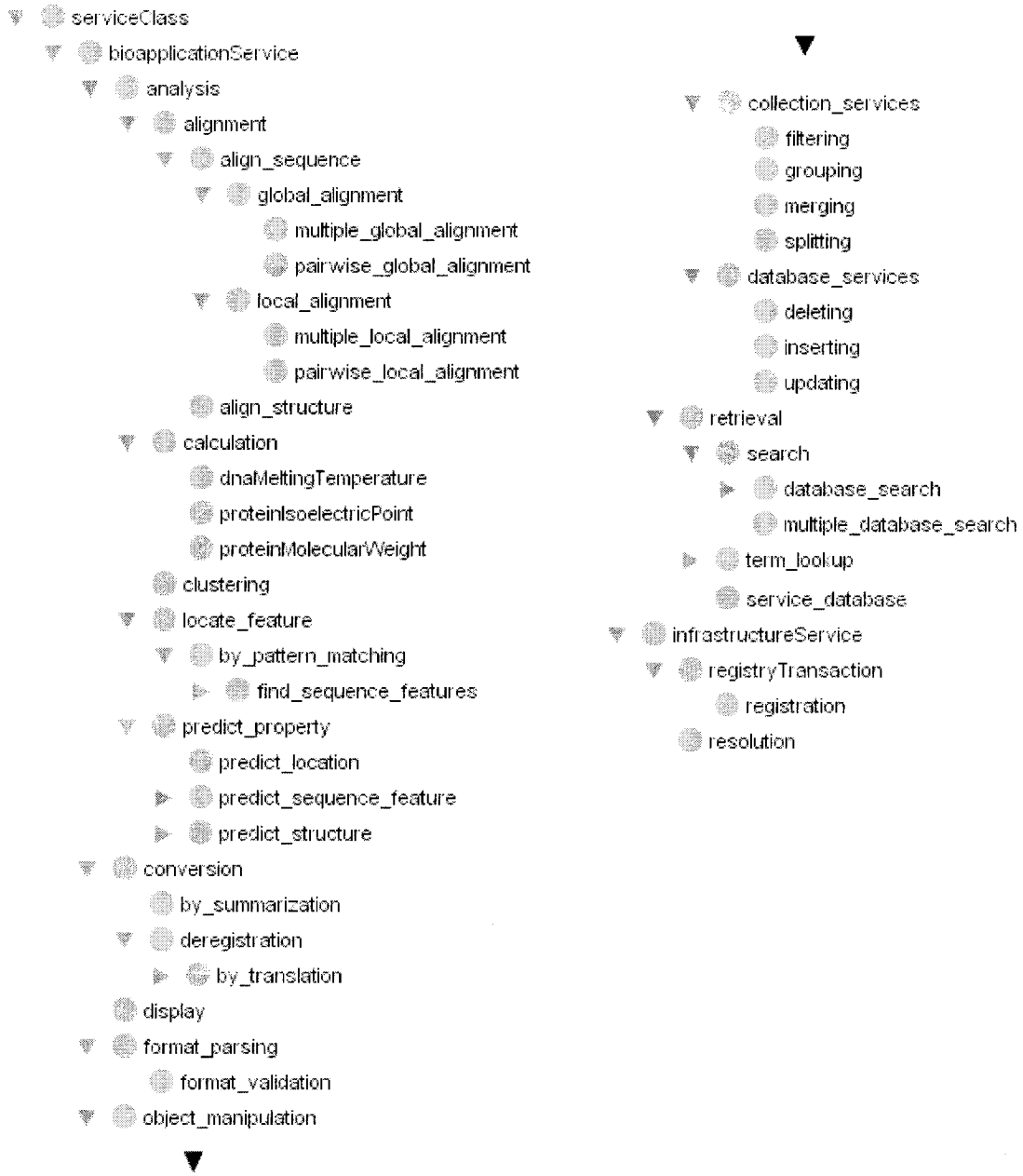


Figure 19: The service ontology

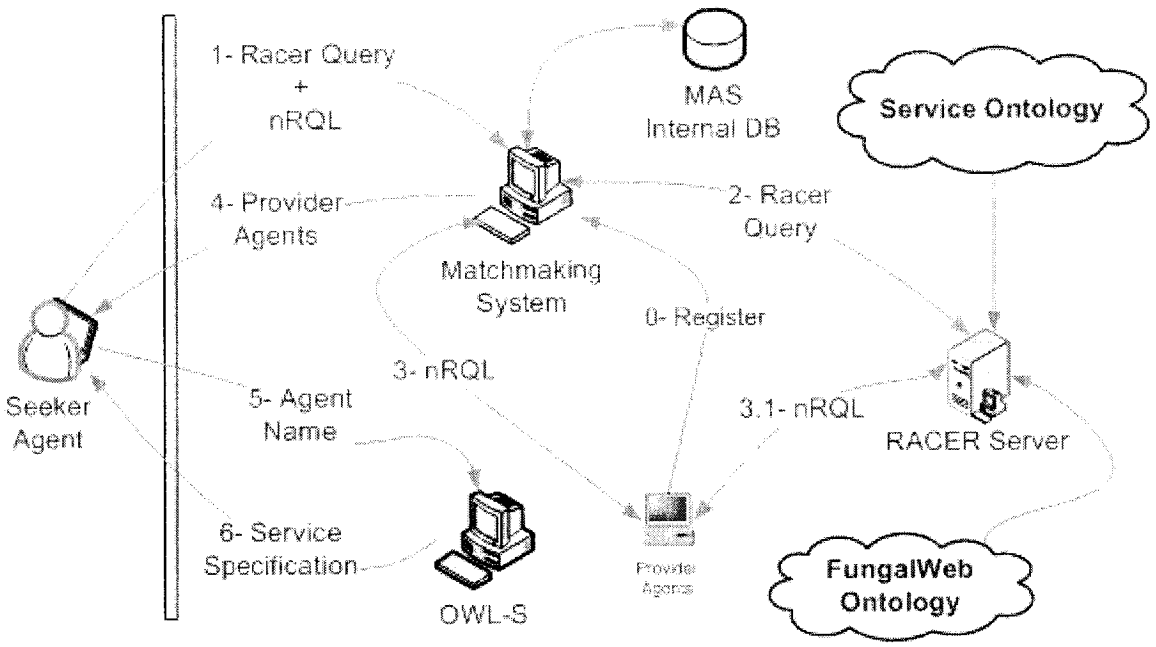


Figure 20: Architecture of the agent system in respect to the provider agent and their accessibility

RACER query in hand, the agent system consults the service ontology and its internal database to get a list of registered provider agents. Next, the matchmaker sends the nRQL query to the provider agents in the list to find out which ones are able to fulfill the seeker's need. The provider agents run the nRQL query against the FungalWeb ontology to semantically understand what exactly the seeker is looking for and they reply back positive to the matchmaker if applicable. After that, the matchmaker sends the list of approved agents to the seeker. The seeker then consults the OWL-S component to find the appropriate way of querying each provider agent and then it contacts the provider agents.

Moreover, Figure 21 shows the agent system architecture in a broader context. The seeker agents are out of the scope of the agent system, and they only share the service ontology and the FungalWeb ontology with the agent system. The agents in the system are categorized in the three types of:

- ETL agents that update the data warehouse. They retrieve the data from the

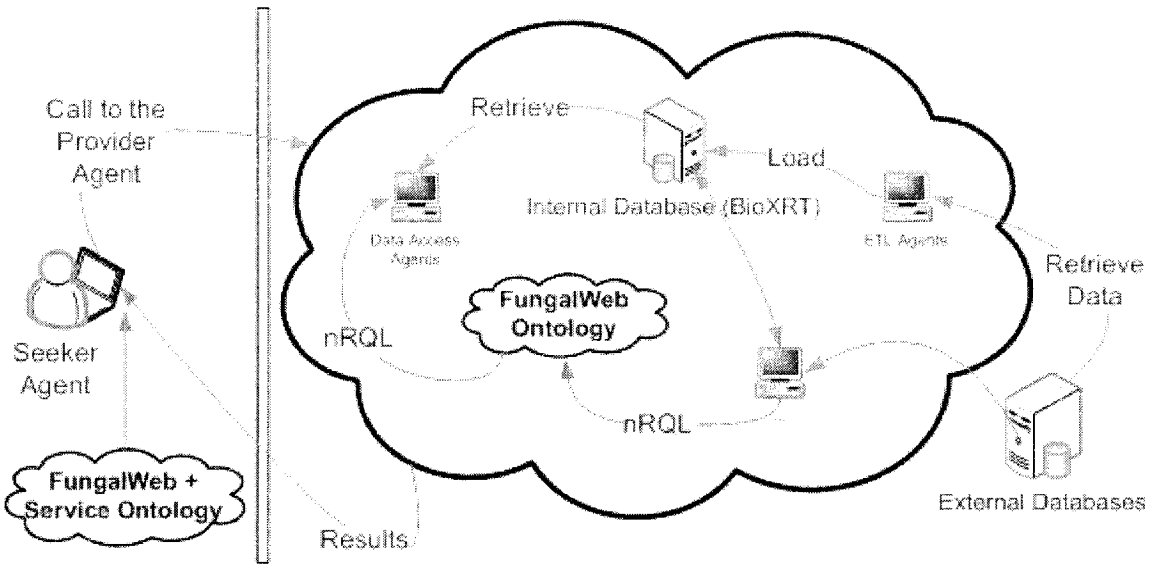


Figure 21: Architecture of the agent system in a broader context

data warehouse, transform the data to the data warehouses format, and load the transformed data into the data warehouse.

- Data Access agents that provide access to the data in the data warehouse. They use the FungalWeb ontology as the way to understand in what exact data the seeker is interested.

- Other agents that range from computational agents to external source agents. The computational agents may provide access to biological computational service as a transparent database, and the external source agents have been targeted on providing access to the data is not integrated.

4.3 An Example Provider Agent

As an example of provider agents in the agent system we explain how geneAgent fetches data from external sources and loads it into the data warehouse. The geneAgent is chosen because it is an updater agent, and it explains how new updater agents should be developed.

The design and implementation of the geneAgent have two sides. One side is the steps should be taken for DECAF and Matchmaker Agent System compatibility and the other side is the steps should be taken to load the data into data warehouse.

4.3.1 DECAF and Matchmaker Code

In order to get the geneAgent to work in the DECAF framework it has to have a plan file. Therefore, the first step is designing the agent plan file. As discussed before, provider agents extend the matchmaker system's provider agent plan file, shown in Figure 7. In the case of the geneAgent we do not need to define new tasks in the agent plan, and the whole logic can be developed in the `check` action. Therefore, copying the provider agent's plan file is good enough. Next, the DECAF framework generates the Java code for the agents. The Java code has method bodies for the actions inside the plan file, and the methods need to be populated with appropriate method.

The first method to implement is the `_startup` action. In this method, the agent notifies the matchmaker agent, and registers itself in the system. The code below is the implementation of the `_startup` action for the geneAgent.

```
1  import taems.Agent;
2
3  public class geneAgent_Startup
4  {
5
6      public geneAgent_Startup()
7      {
8      }
9
10     public ProvisionCell Action(LinkedListQ Plist, Agent Local)
11     {
12         KQMLmsg outKQML = new KQMLmsg();
13         outKQML.addFieldValuePair("performative", "achieve");
14         outKQML.addFieldValuePair("sender", Local.getName());
15         outKQML.addFieldValuePair("receiver", "Matchmaker");
16         outKQML.addFieldValuePair("ontology", "Matchmaker");
```

```
        outKQML.addFieldValuePair("language", "DECAF");
        outKQML.addFieldValuePair("content", ":task advertise" +
                " :keywords object_manipulation" +
                " :ontology " + "geneProvider");

        return new ProvisionCell(outKQML.getKQMLString(), "OK");
    }

}
```

The code simply creates a new KQML message and sends it to the matchmaker agent. The message triggers the advertise task of the matchmaker agent, and it provides the task with the appropriate set of parameters. The `task` parameter specifies the `advertise` task of the matchmaker should be invoked. Furthermore, the `keyword` parameter specifies the service of this agent, which is a general concept such as `object_manipulation` in this case. The service concept matches those in the service ontology as shown in Figure 19.

Having registered in the matchmaker, the agent needs to implement the `deeper` action to let the seeker agents understand its service. The primary goal of this method is to confirm the service availability with the matchmaker agent, thus the seeker agents. As discussed before, we use the FungalWeb ontology to agree on the fungal genomics terminology with the seeker agents. That is, the provider agents in this agent system expect to receive an nRQL querying the instances of FungalWeb ontology.

The code below shows the implementation of `deeper` action for the geneAgent.

```
import jracer.*;

public class geneAgent_deeper
{
    String RACER_ADDRESS = "127.0.0.1";
    int RACER_PORT = 8088;
    String Answer = "";

    public geneAgent_deeper()
```

```
10      {
11      }
12
13      public ProvisionCell check(LinkedListQ Plist, Agent Local)
14      {
15          String Count = Util.getValue(Plist, "count");
16          String RQL = Util.getValue(Plist, "RQL");
17          String AgentsNames = Util.getValue(Plist, "AgentsNames");
18          String origiSender = Util.getValue(Plist, "origiSender");
19          String ID = Util.getValue(Plist, "ID");
20          String Decoded_RQL = decode(RQL);
21
22          RacerClient Rclient = new RacerClient(RACER_ADDRESS,
23                  RACER_PORT);
24
25          try
26          {
27              Rclient.openConnection();
28
29              try
30              {
31                  Answer = Rclient.send(Decoded_RQL);
32
33                  if (Answer.matches("NIL"))
34                  {
35                      // This means that there's no Answer for the RQL
36                      Answer = "SORRY";
37                  }
38              }
39              catch (RacerException e)
40              {
41                  e.printStackTrace();
42              }
43
44              Rclient.closeConnection();
45          }
46          catch (Exception e)
47          {
48              e.printStackTrace();
49          }
50
51          Answer = encode(Answer);
52
53          if (Answer.indexOf("http://a.com/ontology#Gene") == -1)
54              Answer = "SORRY";
```

```
55
56          KQMLmsg outKQML = new KQMLmsg();
57          outKQML.addFieldValuePair("performative", "achieve");
58          outKQML.addFieldValuePair("sender", Local.getName());
59          outKQML.addFieldValuePair("receiver", "Matchmaker");
60          outKQML.addFieldValuePair("ontology", "Matchmaker");
61          outKQML.addFieldValuePair("language", "DECAF");
62          outKQML.addFieldValuePair("content", ":task deeper" +
63                  " :count " + Count +
64                  " :RQL " + RQL +
65                  " :AgentsNames " + AgentsNames +
66                  " :answer " + Answer +
67                  " :origiSender " + origiSender +
68                  " :ID " + ID);
69
70          return new ProvisionCell(outKQML.getKQMLString(), "OK");
71      }
72
73      public String encode(String query)
74      {
75          char oldch = '(';
76          char newch = '<';
77
78          query = query.replace(oldch, newch);
79          oldch = ')';
80          newch = '>';
81          query = query.replace(oldch, newch);
82          oldch = ':';
83          newch = '$';
84          query = query.replace(oldch, newch);
85
86          return query;
87      }
88
89      public String decode(String query)
90      {
91          char oldch = '(';
92          char newch = '<';
93
94          query = query.replace(newch, oldch);
95          oldch = ')';
96          newch = '>';
97          query = query.replace(newch, oldch);
98          oldch = ':';
99          newch = '$';
```

```
100             query = query.replace(newch, oldch);
101
102             return query;
103         }
104     }
```

In the code, first a connection is made to the Racer server, and the nRQL query is sent to it. Racer server runs the query and returns back the results to the agent. Note that the Racer should load the FungalWeb ontology first or it will not be able to return any result.

The code checks the returned answer. The agent's answer is sorry if there has been no answer for the nRQL in the FungalWeb ontology. If not, the agent check if the user query covers the `gene` concept, and if so it returns a positive answer to the matchmaker.

Besides the process of confirming user request, note that the `Count`, `AgentNames`, `RQL`, `origiSender`, and `ID` fields in the KQML message play the role of matchmaker's memory of the seeker agent's request. The matchmaker creates a list of agents based in the ConceptQuery it receives from the seeker agent. Then it sends the nRQL to each of the agents in the list, but it does not keep the list in its memory. This makes the matchmaker a stateless agent, and it makes the matchmaker agent robust. After this step, the geneAgent is good to go and it can be run within the matchmaker agent system. However, the `activate` action should be implemented to provide the actual service.

4.3.2 ETL Code

The service of the geneAgent is to fetch FASTA files for a gene and load it into the database. For this particular agent this is done in two steps.

The first step downloads a remote file and uses the FastaGeneConverter transformer class to change the FASTA format to the tab delimited format. The FastaGeneConverter gets a FASTA file and uses the BioJava library (`http://biojava.org/`) to interpret the FASTA file and turn it into the destination format. The code assures the new IDs that are assigned to the genes are unique. The code for the FastaGeneConverter is too big to be here; however, it can be downloaded from `http://www.cs.concordia.ca/~f_kohant/thesis/tsource.zip`.

After the tab delimited file is ready, the agent needs to process the content of the records and modify them. In this particular example, the agent needs to find the GenBank ID of the retrieved genes. At this stage, the geneAgent uses Clover ETL to convert the data. Clover ETL is a Java based ETL framework. The use of such frameworks makes the agent system code standard and easy to understand. However, you may have noticed that the Clover ETL has not been used for the first step. The reason is that for the moment the Clover ETL does not support raw text files as input in a straight way. Later releases may support such situations and are appropriate for the first step.

In order to use the Clover ETL framework, an ETL graph should be designed. Clover ETL comes with a graph designer as a plugin for Eclipse IDE. Figure 22 shows the graph used for the geneAgent. The graph for converting new gene files consists of two paths that are merged together before constructing the output. The first path is responsible for formating new records and the second path is responsible for removing duplicate entries.

Having the appropriate ETL graph in hand, the agent needs to load and run the graph. The following code snippet loads the graph and runs the process in the geneAgent. One problem with the current version of the Clover ETL is that it does not have a straightforward way of setting values in the graph. The only possible way is the GUI, and it does not work for our agent system. Therefore, we tend to modify

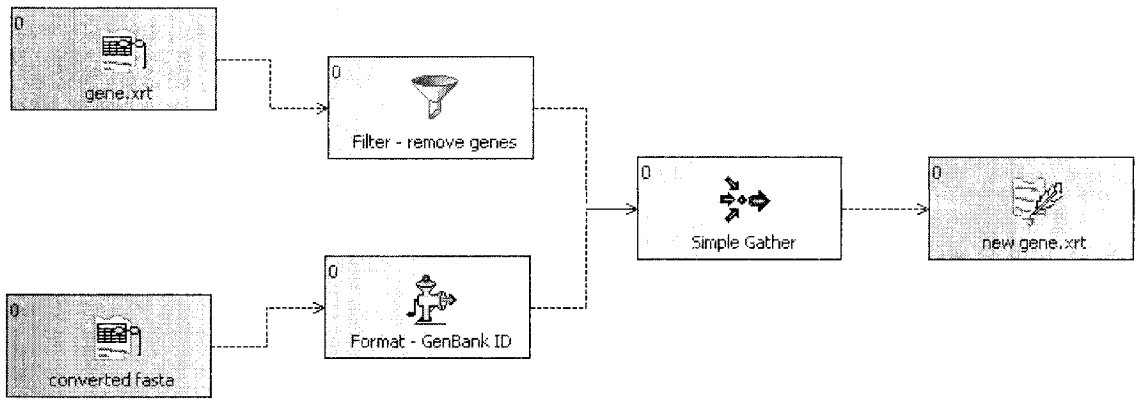


Figure 22: The ETL graph for converting Gene records

the graph XML file before loading it. Clover ETL graph files are standard XML files, and their manipulation is trivial.

```
1         Defaults.init();
2         ComponentFactory.init();
3
4         TransformationGraph graph = new TransformationGraph();
5         TransformationGraphXMLReaderWriter graphReader =
6                 new TransformationGraphXMLReaderWriter(graph);
7
8         setInputFileName("geneAgent.grf", "converted fasta",
9                          "somefastafile.fasta");
10
11        setTaxID("geneAgent.grf", "Filter - remove genes",
12                         "13432");
13        InputStream in = new FileInputStream("geneAgent.grf");
14
15        if (!graphReader.read(in))
16        {
17            System.err.println("Could not load the graph!");
18            return;
19        }
20
21        if (!graph.init())
22        {
23            System.err.println("Graph initialization failed!");
24            return;
25        }
26
```

```
27          graph.run();
```

Upon execution of the graph the two paths start processing their inputs. In the first path the data in the gene.xrt is read and passed to a filter component. The filter component examines if the TaxID column of the record is different from the organism that is currently being processed. It also checks the source of the data. That is, the combination of TaxID and source fields constructs a unique key for the FASTA files that are taken from a specific source. The following XML snippet shows the filter element in the ETL graph XML file:

```
<Node enabled="enabled" guiHeight="0" guiName="Filter - remove genes"
     guiWidth="0" guiX="288" guiY="83" id="FILTER0" type="FILTER">
    <attr name="filterExpression">TaxID!=4556
                or source!=MIT Broad Institute</attr>
</Node>
```

The records that satisfy the `filterExpression` condition pass through this component and reach the merger component. The merger component is responsible for gathering the data from the first path and the second path.

The second path of the graph reads the converted fasta file and pass the records to the GenBank ID transformer. GenBank ID transformer is a custom transformer that submits a gene sequence to the NCBI Blast web site and extracts the GenBank IDs from the results. The code below shows the code for GeneGenBankIDTransformer.java transformer.

```
1  package org.concordia.cs.fungalweb.data.etl.transformers;
2
3  import org.jetel.component.DataRecordTransform;
4  import org.jetel.data.DataRecord;
5  import org.jetel.data.GetVal;
6  import org.jetel.data.SetVal;
7  import org.concordia.cs.fungalweb.ncbi.qblast.ResultListener;
```

```
import org.concordia.cs.fungalweb.ncbi.qblast.NCBIRunner;
import org.concordia.cs.fungalweb.ncbi.qblast.NCBIBlastScheduler;
import org.concordia.cs.fungalweb.ncbi.qblast.result.Hit;

public class GeneGenBankIDTransformer extends
        DataRecordTransform implements ResultListener
{
    public boolean transform(DataRecord[] source, DataRecord[] target)
    {
        for (int i = 0; i < source.length; i++)
        {
            DataRecord source_data_record = source[i];
            DataRecord target_data_record = target[i];

            String seq = GetVal.getString(source_data_record,
                    "sequence");

            NCBIBlastScheduler.getInstance().queueBlast(seq, null,
                    this, "blastn");

            while (!resultReturned)
                try
                {
                    wait(2000);
                }
                catch (InterruptedException e)
                {
                    e.printStackTrace();
                }

            // It is possible that no match is found for the gene,
            // then we have nothing to do here.
            if (hit == null)
            {
                return true;
            }

            // In the case OtherIDs is null from the original source
            // it istaken from the NCBI
            if (GetVal.getString(source_data_record, "OtherIDs").
                    trim().length() == 0)
                SetVal.setString(target_data_record, "OtherIDs",
                        hit.getDefinition());

            SetVal.setString(target_data_record, "GeneID",
```

```
53                      hit.getId());
54          }
55
56          return true;
57      }
58
59      public void resultsReturned(NCBIRunner runner)
60      {
61          hit = runner.getHit();
62          resultReturned = true;
63      }
64
65      protected boolean resultReturned = false;
66      protected Hit hit;
67  }
```

The code uses the NCBI Blast Java Toolkit to schedule a Blast request on the NCBI web site. The NCBI Blast Java Toolkit, developed during this project, is a Java wrapper for the NCBI Blast web API. The toolkit is able to schedule all of the Blast algorithms on the NCBI web site, and it provides means of tweaking the parameters. Furthermore, the toolkit uses multi threading techniques to check server for the availability of results. The toolkit has also been tuned so that it does not violate the usage policies of the NCBI server.

The transformer sets the value of two fields. First, it sets the value of the the `OtherGeneIds` fields if it is empty. The FASTA transformer fills this field with the key it extracts from the FASTA file. However, not all of the FASTA files include keys for their sequences, and this value can be filled with the description field in the Blast result. The second field the transformer sets is the `GeneID` field which should be filled with the GenBank ID returned in the Blast result.

After all of the records are processed by this transformer, which can take fairly a long time, their are redirected to a merger component, and they are merged with the filtered genes.

Finally, the merger component merges the results and send them to the tab delimited file writer. Till this point the geneAgent is done. The new XRT file is later loaded into the data warehouse using the BioXRT's bulk loader script.

Chapter 5

Discussion and Conclusion

In this chapter we compare our agent system and data warehouse with other related works to magnify the differences and explain the current trends in the domain. Moreover, the chapter finishes by summarizing the work, stating our contribution, and evaluating the data warehouse.

5.1 Related Work

As explained in the Chapter 3, the integrated database systems are classified in the three types of portal oriented systems, mediators, and data warehouses. A part of this research is a data warehouse. This section brings an example of a system for each of these types in order to compare the types and discusses the pros and cons of each system against the data warehouse developed in this research. Taverna has also been explained here as a new initiative in accessing biological services; therefore it is a comparison to the service oriented aspect of this research.

5.1.1 SRS

Sequence Retrieval System (SRS) [EA92] is an example of a commercial portal oriented system. What SRS does is that it does not offload the data from external databases to its internal database, but it builds an internal repository in which it keeps indexes of the original data. In addition to that, SRS stores indexes of relations between the external data. This gives SRS the power to offer a reasonable suit of query tools to its users.

The philosophy behind the SRS is that it models the external data in an object-oriented manner. That is, it defines a class for each concept data in the external sources, and the objects of the classes mirror the original data. The classes play the role of the meta data in the system, and they store enough information to enable SRS query the external data and access it. In addition to classes, SRS assigns parsing rules to classes so that the SRS knows how to access the data and interpret it. The SRS also introduces a scripting language, Icarus, for object and rule definition.

One key feature of the SRS is connecting data through indexed links. The indexed links are defined based on the explicit and implicit relation of data in multiple external sources. For example, an entry in the SWISS-PROT database can be linked to the ENZYME database by the EC number (implicit relation), or the same entry can be linked to an entry in PDB by the same accession number it has in both databases (explicit relation). It is possible the links be a part of the SRS queries as they are bi-directional, are weighted, and can be combined with logical operators to form more interesting queries. For example, "give me all proteins that share InterPro domains with my protein" [ZLAE00] is possible by linking SWISS-PROT to InterPro and back to SWISS-PROT.

The SRS comes with a server which respondes SRS requests, and a web interface which facilitates user queries. The current version of the SRS indexes over 400 databases, some of which has been listed in the Table 6.

Sequence			
	EBML	EMBLNEW	ENSEMBL
	SWISSPROT	SPTREMBL	TREMBLNEW
	SWALL	IMGT	IMGTHLA
	REMTREMBL		
InterPro & Related			
	InterPro	InterProMatches	PROSITEDOC
	PFAMA	PFAMB	PFAMHMM
	PRINTS	NICEDOM	PRODOM
	BLOCKS	PFAMSEED	PROSITE
SeqRelated			
	TAXONOMY	GENETICCODE	EPD
	UTR	UTRSITE	EMESTLIB
	HTG_QSCORE		
TransFac			
	TFSITE	TFFACTOR	TFCELL
	TFGENE	TFMATRIX	TFCLASS
Protein3DStruct			
	PDB	DSSP	HSSP
	FSSP		
Genome			
	HSAGENES	MOUSE2HUMAN	LOCUSLINK
Mapping			
	RHDB	RHEXP	RHMAP
	OMIMMAP	RHPANEL	
Mutations			
	MUTRES	MUTRESSTATUS	OMIM
	OMIMOFFSET	SWISSCHANGE	EMBLCHANGE
	HUMAN	MITBASE	P53LINK
	OMIMALLELE	HUMUT	
SNP			
	MITSNP	dbSNP_Contact	dbSNP_Method
	dbSNP_Population	dbSNP_Publication	dbSNP_Assay
	dbSNP_SNP	dbSNP_PopUse	dbSNP_IndUse
	HGBASE	HGBASE_SUBMITER	SNPLink
Metabolic Pathways			
	PATHWAY	LENZYME	LCOMPOUND
	EMP	MPW	UPATHWAY
	UCOMPOUND	UIMAGEMAP	ENZYME
	BRENDA	UREACTION	UENZYME

Table 6: List of databases accessible through the SRS [ZLAE00]

Advanced Features

The SRS provides the common features of a portal system. However, it also offers some advanced features that are unique in its kind. Few examples of such features are [ZLAE00]:

- Multiple subentries: Looking at data as a flat set of entries limits the schema specification, therefore query abilities. As a result, the SRS allows definition of multiple subentries per entry within the system. Subentries are logically independant concepts of the domain that elaborate an existing concept. An example of subentries can be the elements of Feature Tables in sequence databases such as SWISS-PROT [ZLAE00]. Publication references are another relevant example.

- Virtual data fields: The SRS employs an on-the-fly algorithm to retrieve data on demand instead of storing them. This is made possible by introducing virtual data fields which are in essence the place holder of the original data. Virtual data fields are methods which lazy parse the source data upon receipt of a data access request. The use of virtual data fields enables runtime customization of the original data based on the view layer.

- Composite views: Composite views create dynamic views of result sets when they are from multiple sources and there needs to be a nice way of viewing them. For example, a gene may need a customized view to be shown with its metabolic pathways.

- Integration of data analysis application: Not only does the SRS provide access to remote data as if they were local, but the SRS also supports access to computational services. This way, the result of computational services can be treated as a dynamic database, and users can pipeline the results of such services in

their query results integration. CLUSTALW [HTHG94] is an example of such service which is integrated in the SRS.

Programing Interface

As an enhancement in the new version of SRS (SRS6), "SRS Objects" has been made available as means of accessing SRS services in the programing languages. This enables C++, Java, Phyton, and Perl application developers to access SRS services through native APIs. This way, developing a customized application is a preferred way unless the provided web interface fits. The SRS people have used the programing interface to provide a web interface and a web service version of their InterProScan service [PSK+05].

In addition to the programing APIs, the SRS provides CORBA access to its server. This way, a distributed network of applications can communicate to the SRS server and perform their biological queries.

Discussion

Portal oriented systems are different from data warehouse systems in the sense they do not store the data and they do not have a schema for the data. The former causes portal oriented systems to be slower than warehouse oriented systems, and the latter limits their querying power.

Portal oriented systems redirect their users to the original source; therefore, accessing the data is still slow and users need to understand the data in the original form. In addtion, portal oriented systems are not as useful as data warehouse systems are for computer programs as computer programs need to follow the links to the original data to access the content. Although SRS has proxy objects to provide transparent access to original data, it does not improve the performance as still there is a delay to fetch the data.

In addition, SRS does not function properly if the original data's schema changes. That is, the proxy objects' code and the customized rendering components' code is dependent to the original data's schema, and they need to change when the original schema changes.

Furthermore, portal oriented systems can not offer enhanced cross database search as they do not have any schema of the integrated data. On the contrary, data warehouse oriented systems have a schema of their integrated data and they offer enhanced querying over their integrated data.

Despite all above, portal oriented systems are fairly flexible in integrating new data sources. That is, they do not need to transform the data and they do not need to integrate them. Moreover, they always present the latest data as they redirect requests to the original source.

5.1.2 TAMBIS

The Transparent Access to Multiple Bioinformatics Information Sources (TAMBIS) is a mediator oriented system that provides transparent access to biological sources. Therefore, TAMBIS [BBB+98] is the bridge between users and external sources and it provides data source transparency by introducing a collection of biological terminology and a mapping from those terminologies to data in external sources. In order to do this, TAMBIS employs the classical mediator/wrapper architecture [Wie92], as illustrated in Figure 23.

The first layer in the architecture is the knowledge base of biological terms and concepts, and a user interface. The user interface provides enough facilities for users to combine the biological concepts from the knowledge base and form their queries. The second layer is the mediator layer which identifies which sources should be contacted to answer the given query, and it also translates the query to the proper source-dependent destination external sources. The third layer is the wrapper over the

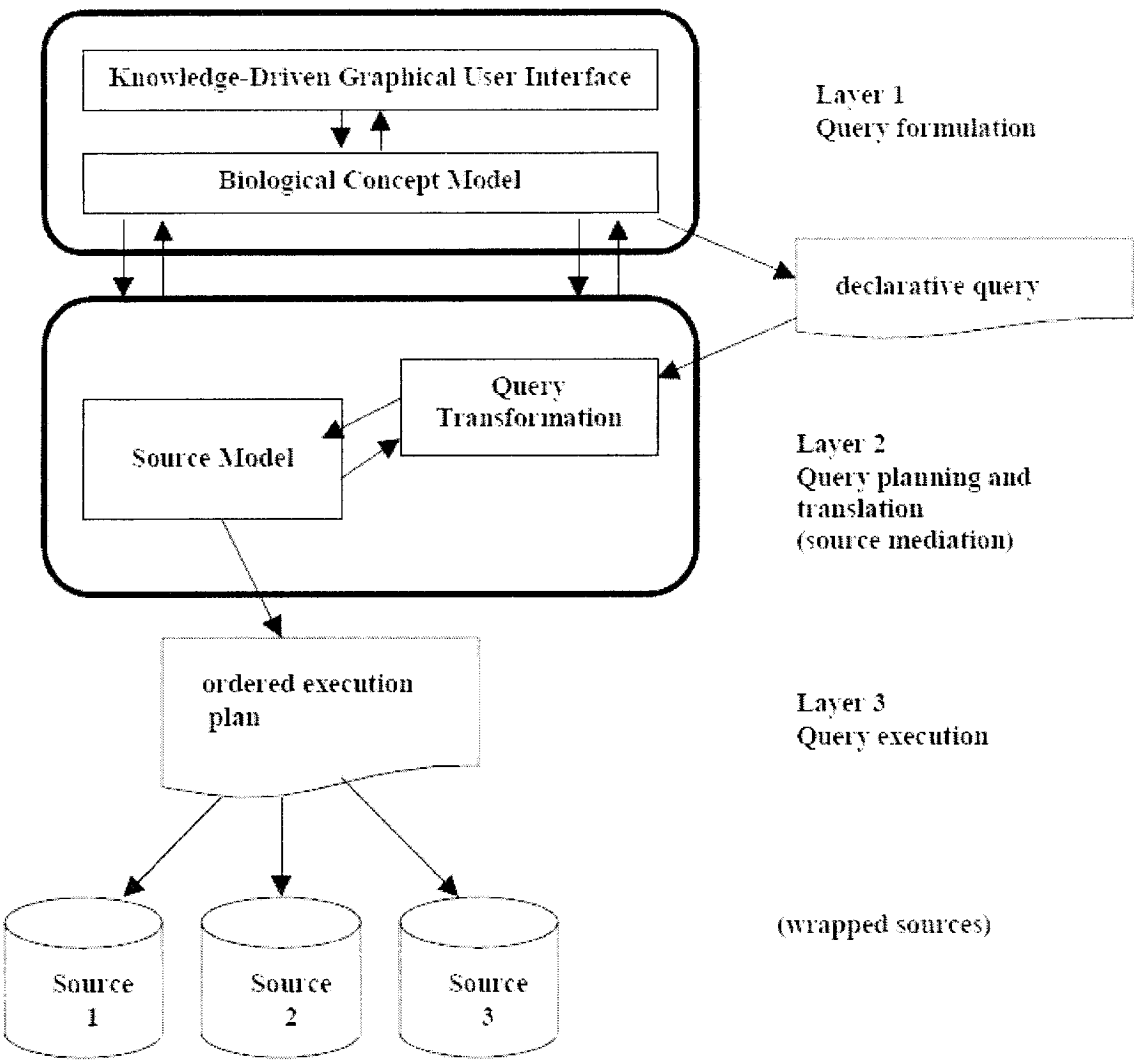


Figure 23: TAMBIS three layer mediator/wrapper architecture, [BBB+98]

external data sources. Wrappers provide data structure and communication tools for the external sources.

The three layers of TAMBIS architecture rely on the existence of five main components underneath the architecture of the system:

- The biological Concept Model
- The graphical user interface
- The Source Model

- The Query Transformation Module
- The Query Execution Module

The Biological Concept Model

The TAMBIS biological Concept Model is a set of biological terms that can be used as a standard language for expressing and communicating ideas between people, and it has enough terms to cover protein and nucleic acids, their component parts and their structures, biological functions and processes, tissues, and taxonomy.

One decent feature of the terminology in the Concept Model is that it is compositional. That is, the basic terms can be recursively combined together to form composite terms. For example, the three terms 'Motif', 'isComponentOf', and 'Protein' can be combined together to form the composite term 'Motif which isComponentOf Protein'. Furthermore, the composite term 'Motif which isComponentOf Protein and hasFunction Hydrolase' can be formed by combining the previous composite term, the terms 'hasFunction', and 'Hydrolase'.

The other feature of the TAMBIS Concept Model is its support of the subsumption relations, also known as 'isa' relations. As an instance, ProteinSequence 'isa' Sequence in the Concept Model.

The two above feature and more are available because TAMBIS uses GRAIL DL language [RBG+97] to define its Concept Model. The GRAIL DL, developed in the Manchester University, is an 'isa' language which allows definition of compositional and recursive models. Figure 24 shows a fragment of the GRAIL representation of the TAMBIS Concept Model. In the Figure, 'Motif' is the main term and it is combined with the 'isComponentOf' term.

TAMBIS uses the Concept Model to define a data schema for external data sources, as a language for constructing user queries, to enhance user interaction with its graphical user interface, and finally to integrate data from different external data

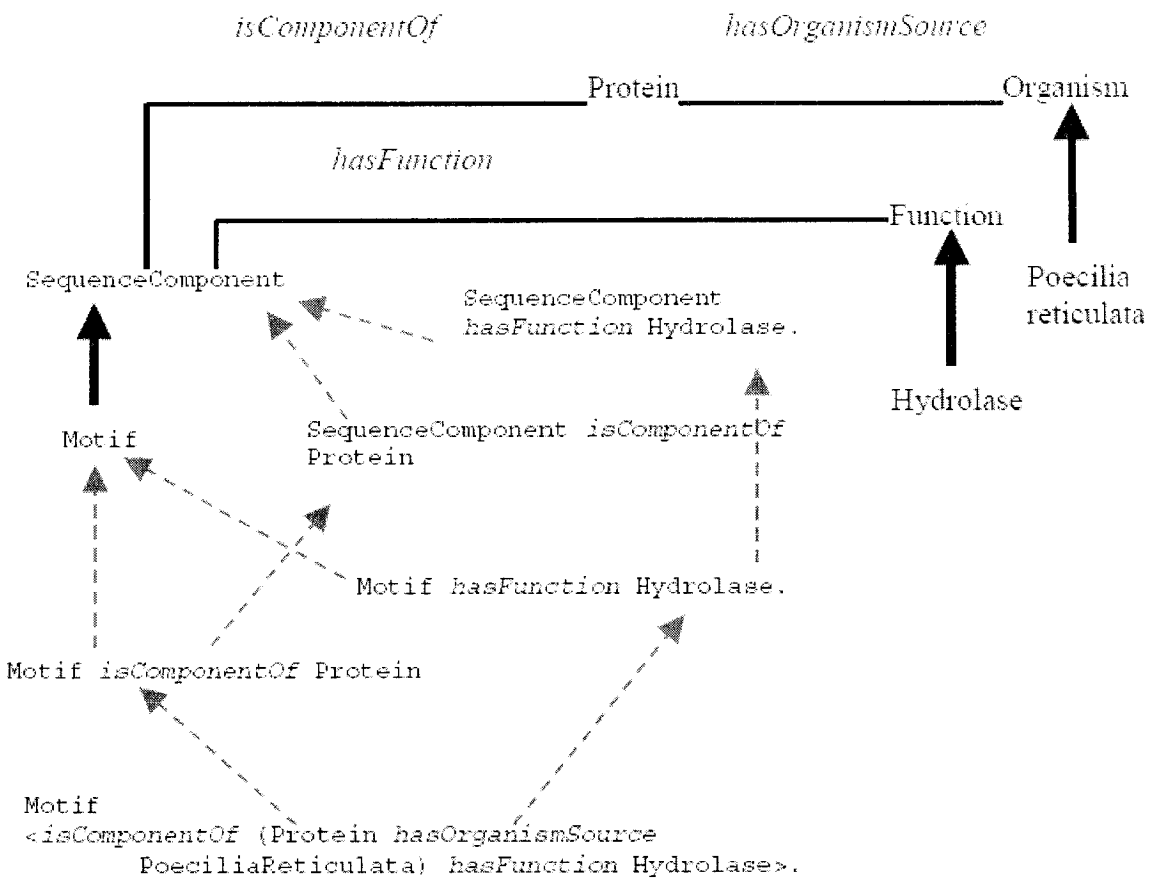


Figure 24: The GRAIL representation of the TAMSIB Concept Model [BBB+98]

source.

The User Interface

As described earlier, the Concept Model terms are used to form user queries, and Concept Model terms are defined in the GRAIL DL language. Therefore, biologists need to learn GRAIL in order to use TAMBIS. However, learning a computer language is an unnecessary overhead for biologists. To remove this overhead TAMBIS comes with a form based graphical user interface that enables constructing user queries by choosing terms from the Concept Model.

As illustrated in Figure 25, the user interface provides a term navigator by which users can look for a term and select it, and they are provided with the related terms

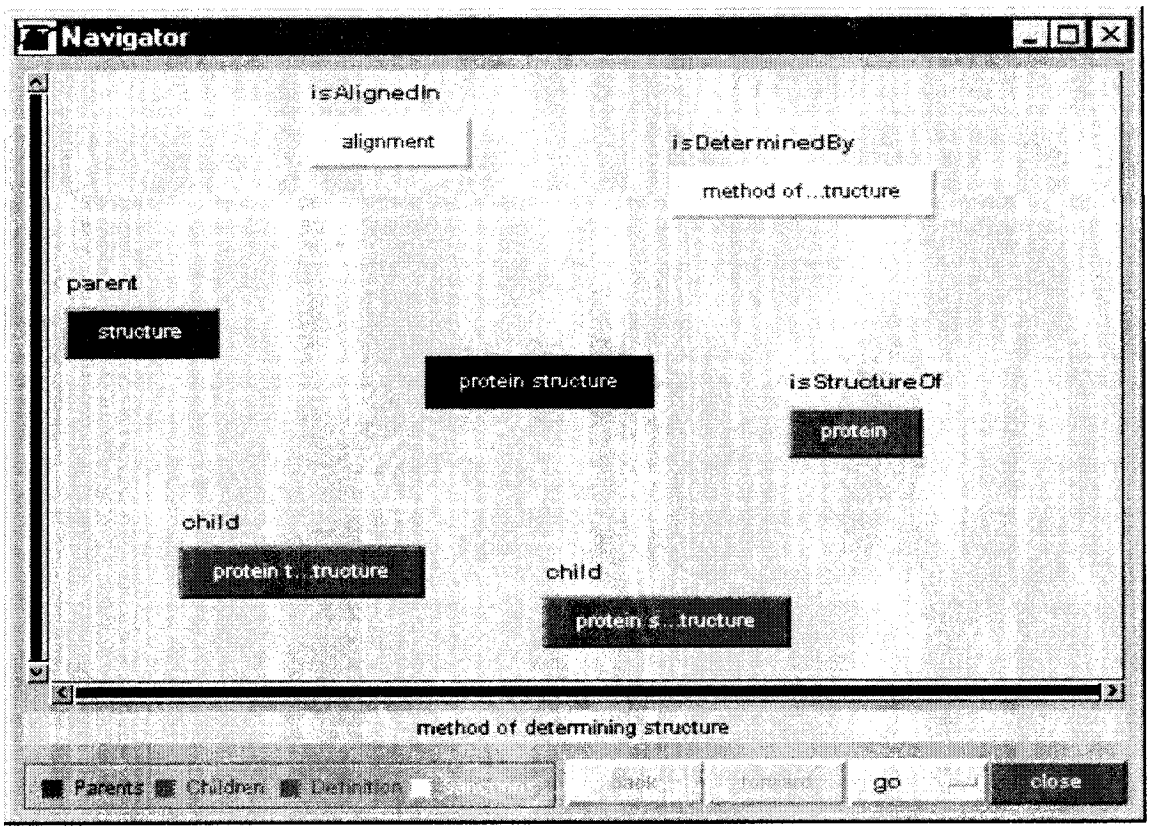


Figure 25: The TAMBIS Graphical User Inteface – The navigator screen with protein structure in the middle, [BBB+98]

to the selected term. In the Figure 25, the selected term, protein structure, is shown in the center of the navigator and all related terms are rendered around the main term. By clicking on the related terms, the user goes further in details and constructs a more complex query.

In each step of constructing the query the user has the option to specify details of the selected term to enrich the query. Figure 26 shows a screen shot of the GUI by which the user is adding details to the criteria of the motif term. The GUI consults the Concept Model to gives the user appropriate suggestions. For example, in Figure 26 the GUI suggests that a motif can be combined with either a 'protein' or a 'nucleic acid' with the 'isComponentOf' term. Having selected 'protein', the user constructs the query 'find all protein motifs'.

Finally, the GUI supports query maintenance during which the user is able to go

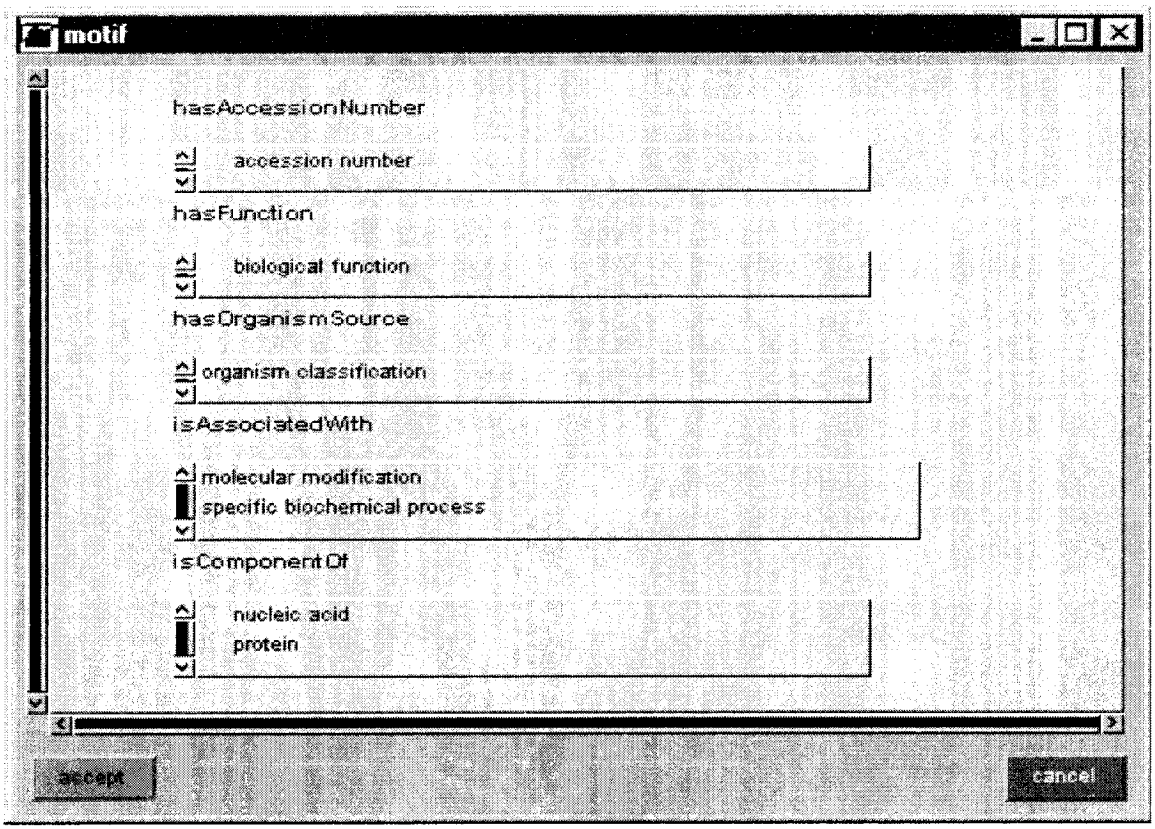


Figure 26: The TAMBIS Graphical User Interface – A screen shot of the GUI when the user is setting query attributes for the motif term, [BBB+98]

back in each step and add more details or remove some terms.

Query Planning and Translation

The TAMBIS user interface helps users ask their questions in terms of the GRAIL language. However, the GRAIL language has no knowledge of the external sources, and the query itself can not be used to retrieve data from them. Therefore, TAMBIS needs a transformer layer that transforms and redirects GRAIL queries to the appropriate external sources. The query planning and translation does this job in the TAMBIS.

The query planning and translation does its job by transforming the given GRAIL query to a Collection Programming Language (CPL) [Won95] execution plan. The CPL is a functional language that supports manipulation of complex data types such

as lists and sets, which are suitable data structures for the biological data. CPL has been extended by some tools such as BioKliesli [BDH+95] to add support for biological data. In TAMBIS, the CPL and BioKliesli combination plays the role of a multiple database language and the support for sending queries to the external sources and pipelining the results.

The query planning and translation phase is comprised of three steps. The first step is to transform the GRAIL query into a Query Internal Form (QIF). This transformation is suggested because the nested structure of the GRAIL queries introduces an implication on the evaluation order of concepts. Therefore, the GRAIL query should be transformed to an unnested form, the QIF. The QIF is a list of tuples (Base, Variable, Criteria, Cost, Cardinality) each of which is a part of the GRAIL query. The Base in the tuple is the term from the GRAIL language. The Variable is the name of the variable to store the results of this part of the GRAIL query. The Criteria is the criteria that the user has set for the term in the GUI. The Cost is an estimate cost of running this part of the GRAIL query, and it is computed by the program. The Cardinality is the size of the result set, and it is also set by the program. As an example, the user is querying 'find all motifs in Poecilia reticulata (guppy) proteins'. The GRAIL form of the query will be 'Motif which isComponentOf (Protein which hasOrganismSource PoeciliaReticulata)', and the QIF form of the GRAIL query will be '[(Motif, Motif-1, [(isComponentOf Protein, Protein-1)], -1, 1), (Protein, Protein-1, [(hasSourceOrganism PoeciliaReticulata, null)], -1, -1)]'.

The next step of the query planning and translation is the query planning. The query planner scans QIF to find the most promising query parts to run, and it repeats till it find the answer to all query parts. This step of query planning and translation takes CPL functions and the availability of external sources into consideration and decides which CPL functions should be used to assemble the results of the query.

The last step of the query planning and translation is the code generation. This

step takes the results of the last step and generates the CPL code that runs the user query. For example, the generated CPL code for the previous user query looks like:

```
{Motif-1|
\Protein-1<get-sp-entry-by-os(''POECILIA+RETICULATA''),
Motif-1<-do-prosite-scan-by-entry-rec(Protein-1)}
```

After the CPL code is generated, it is run and the results are rendered as a HTML page.

Discussion

Mediator oriented systems do not store the original data; therefore, they are slower than data warehouse oriented systems. That is, mediator oriented systems need to go to the original source to fetch the data, and they need to wait for all the sources to respond before they can integrate the results and give it back to the user.

In addition, mediator oriented systems are dependent to the data schema in the original source, and they need to change their schema and code to accept changes in the original data sources' schema. On the contrary data warehouse oriented systems are independent of the the original data schema as soon as they store the data according to their schema.

Despite the above, mediator oriented systems always present the latest data from the original sources as opposed to the data warehouse systems which present the data they have stored.

5.1.3 Biozon

Biozon, Golan Yona et al. [BY06b], is an ongoing research in the computer science department of the Cornell University. Biozon is a knowledge base for biological information such as proteins, structures, domain families, protein-protein interactions,

and cellular pathways. In addition, not only does Biozon keep this information but also Biozon integrates them by establishing their relationship. Thus, the problem of linking similar data does not exist in the Biozon's database.

Moreover, Biozon is a warehouse oriented system that keeps its data in a graph, where biological entities are nodes of the graph and their relations are the edges. Using graphs, Biozon has been able to offer powerful search facilities to its users such as complex queries and fuzzy searches. These searches will be discussed in more depth in this section.

In addition, in order to make Biozon publicly available, Biozon is accessible via `http:\\www.biozon.org`, where online users can:

- Browse and navigate through biological entities and see their profile pages, where each entity is shown in its biological context.
- Form and run complex and fuzzy queries, which will be discussed in more details later in this report.
- Rank their search results so that they get closer to what they are looking for.
- Export and distribute their research result using their online accounts.
- Use online analysis tools.

The Data Model

As described earlier, Biozon stores its data in a graph where biological entities are graph nodes and their relations are the edges. Figure 27 shows an example of a subgraph in which a protein is shown with its corresponding sequence, the DNA sequence that encodes it, and the interactions it is involved in.

Although a graph is a good way to store data that are highly interconnected and mutually related, graphs per se do not have a data schema. Thus, it is not easy to

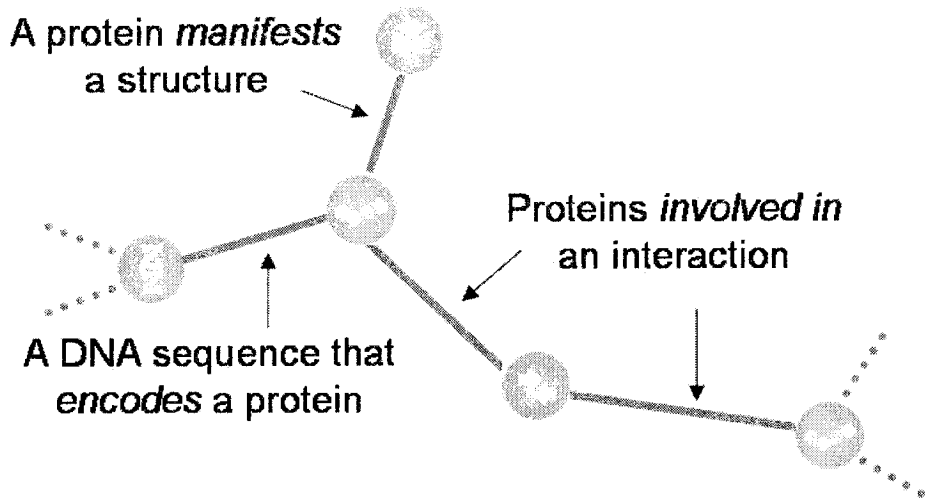


Figure 27: Biozon graph of biological entities

update them and/or integrate new data types into them. However, biological data are being generated at a high speed daily, and new data types are introduced every now and often. Thus, as a major concern, Biozon should be so flexible that it can easily update its graph and incorporate new data types. To solve this problem, Biozon uses a hierarchical data scheme approach to assign type to its graph nodes and edges. That is, each node in the Biozon's graph belongs to a certain class that is in documents' class hierarchy, as shown in Figure 28. Similarly, each edge in the Biozon's graph is of a class node in the relations' class hierarchy, as shown in Figure 29. These class hierarchies feature the inheritance, by which subclasses inherit parent class's attributes. This simplifies maintenance and expandability of typing.

Documents node is the root of documents' class hierarchy which implies the fact that each biological entity in Biozon is a document, some of which has been listed in Table 7. Documents class has three attributes: DocID, Timeline, and Marked. DocID is a unique identifier for the document which is unique in the whole Biozon graph. Timeline is a time tag which shows to where in the Biozon's life time this document belongs. That is, Biozon does not physically remove the old documents, and it tracks to which dataset each document belongs. This may result in a larger

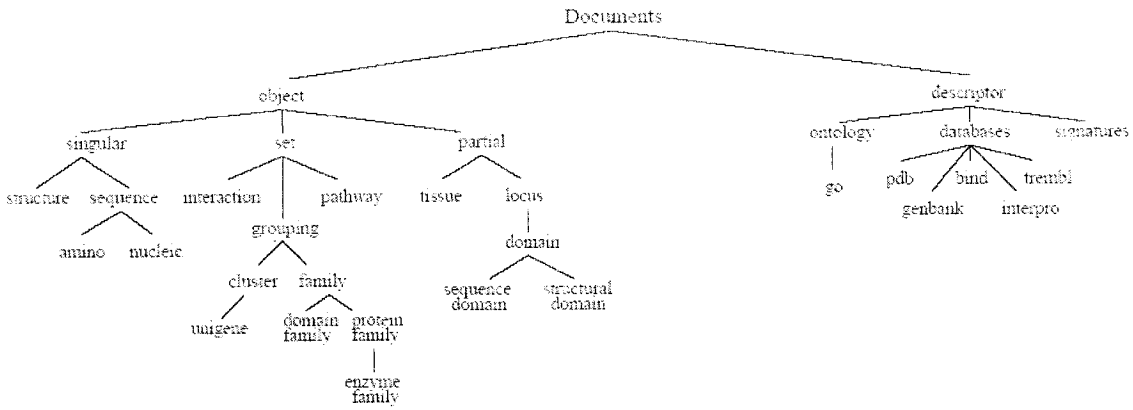


Figure 28: Documents' data scheme hierarchy

graph, but at the same time it gives the users the option to rerun the queries they have run on a certain dataset which has been updated through time, assuring that they will have the same results as they have had at that time. Finally, the Marked attributed is used for detecting deleted and active documents.

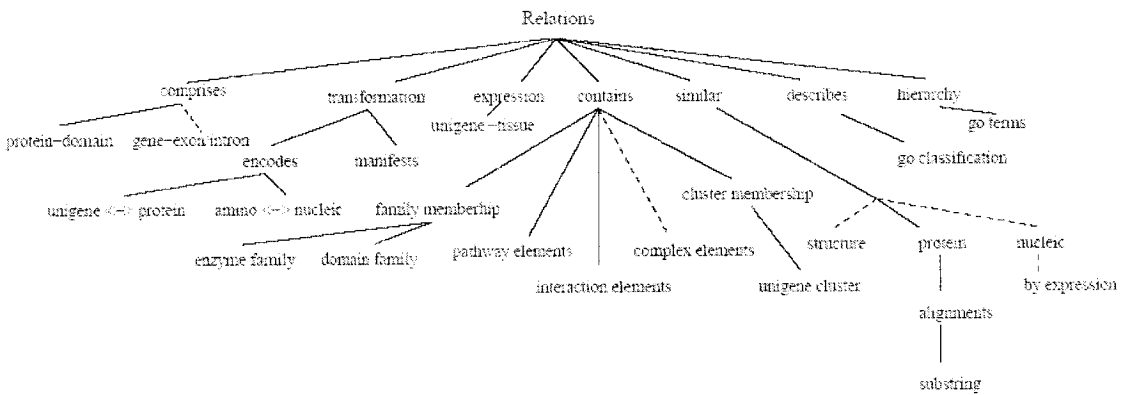


Figure 29: Documents' relation hierarchy

Documents class has two subclasses: object and descriptor. Objects are Biozon's physical entities (such as a sequence), logical entities (such as a domain), or sets (such as a protein family). On the other hand, descriptors describe objects. They contain the information needed to explain what the object is, where from it has been taken, and so on. It is possible that an object has relation with many descriptors.

Document Type	Representation	Atomic Units
protein sequence	string	amino acids
nucleic acid sequence	string	nucleic acids
protein family	set	proteins
pathway	set	protein families
domain	ordered pair	sequence coordinates
domain family	set	domains
interaction	set	proteins, nucleic acids
descriptor	text	characters
structure	list	3D coordinates
unigenere cluster	set	nucleic acids (ETSs)

Table 7: Biozon document types

As described earlier, Biozon identifies the type of its entity relations by assigning a class to each relation from relations' class hierarchy. That is, it is not only important that two entities are related but it is important that how they are related. This will give more flexibility in forming complex queries which will be discussed later. Moreover, the relations in the hierarchy are defined based on the object they refer. For example, similarity relation is between two proteins, and it defines which one is similar to which. Similar to the documents' class hierarchy, the relations' class hierarchy starts with the Relations node, and it subclasses to other classes based on the diversity of the relations in the Biozon datasets. Table 8 shows some of the relations that are currently used in Biozon.

Not only do relations give meaning to the way two entities are related but also they have attributes that give more depth to the entities' relation. For example, when a protein is described by a GO number, and the mapping to the GO number has some evidences, those evidences are attributes of the 'describes.go' relation that the protein has with the GO number. For the moment, Biozon does not support querying these relation's attributes, but that is a nice feature to add in the future.

Relation Type	Referring Document	Referred Document
manifests	protein	structure
describes	descriptor	any object
encodes.nucleic	nucleic acid	protein
encodes.unigene	unigene cluster	protein
similarity	protein	protein
contains.unigene	unigene cluster	nucleic acid
contains.interaction	interaction	protein, DNA
contains.pathway	pathway	enzyme family
contains.enzyme-family	enzyme family	protein
contains.domain-family	domain family	domain
comprises.domain	domain	protein
expresses.unigene	unigene cluster	tissue
hierarchy.go	go term	go term
describes.go	go term	protein

Table 8: Biozon relation types

Datasets

Biozon's internal database contains two kinds of data: source data and derived data. Source data is the data that has been taken from external databases such as SWISS-Prot and KEGG. On the other hand, derived data is the data they have computed internally from the source data. As of now, Biozon's derived data is similarities between proteins sequences, similarities between protein structures, and similarities based on gene expression data. Table 9 shows a list of Biozon's database content by August 2005.

One of the challenges that Biozon people should face is that they should compute these derived data when they update their source data. This can be time consuming and can cause consistency problems in Biozon's data warehouse. However, Yona et al. [BY06b] discuss that they implemented protocols to run updates on their internal database in such a way that everything is working after.

Browsing entities in Biozon

For the moment, web interface is the only interface to the Biozon's database. In the web interface, each entity has a profile page in which all the related information

Data Type	NR Record Count	Sources
nucleic acid sequences	42,686,711	GenBank, BIND
protein sequences	2,062,061	UniProt, GenPept, PDB, BIND
protein structures	32,637	PDB
interactions	155,090	BIND, Biozon (predicted), DIP[a]
enzyme families	3,944	UniProt, PIR, GenPept, SCOP
pathways	142	KEGG
unigene clusters	185,543	NCBI
domain families	181,500	InterPro, Biozon(predicted)
sequence alignments	5,000,000,000+	Biozon
structure alignments	8,250,286	Biozon
non-similarity relations	136,972,705	All
descriptor documents	58,176,040	All
words indexed	1,627,747,755	All

Table 9: Biozon database contents

to the selected entity is shown. In details, the profile page shows the other entities that have a relation with the selected entity. In addition, the profile page shows a list of descriptor documents that are in relation with the selected entity. Thus, the profile page shows a broader biological context of the selected entity. That is, the researchers can see the whole information available about the selected entity in one shot. Figure 30 shows a screen shot of the profile page of a protein that is associated with the breast cancer.

As Figure 30(a) shows, the profile page begins with a summary part that contains the alternate names of the entity and the other general information of the entity. Next, it shows the actual physical record behind the entity, which can be shown in a picture, by a sequence, or in a table. The profile page continues with the related descriptor documents. Finally, the profile page ends by listing the related entities of the selected entity. In the case of the example shown in Figure 30(d), the only related entity is the nucleic acid sequence that encodes the selected protein.

Although Biozon shows all the related information in the profile page, Biozon does not load the content of descriptor documents for both performance and UI reasons. First, loading those related information can be expensive while the user may not be

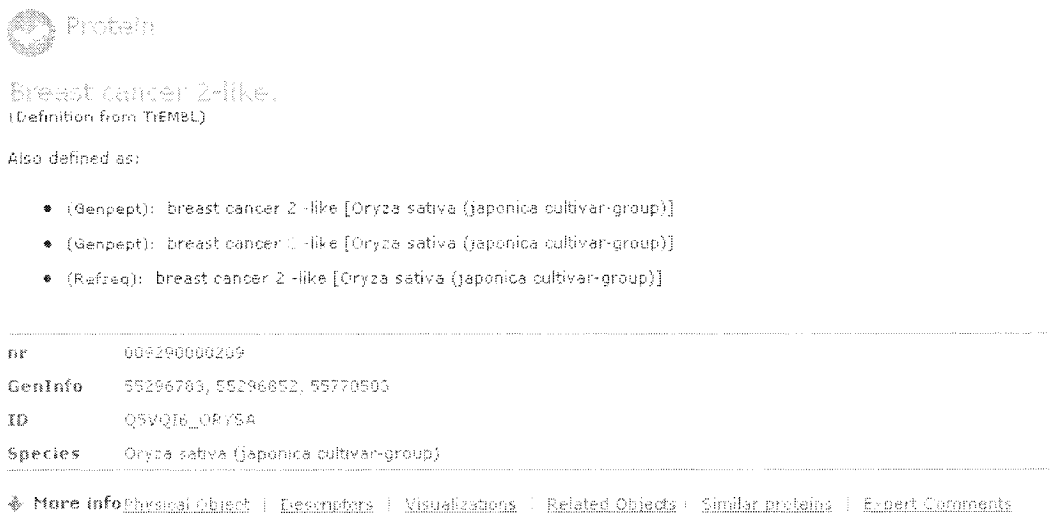
interested in viewing those information. Second, each of those descriptor documents need a special view; For example, a GO entry will need a graph image while a TrEMBL entry fits best in a tabular UI. Showing all these different views at the same time will confuse the user.

Searching Entities in Biozon

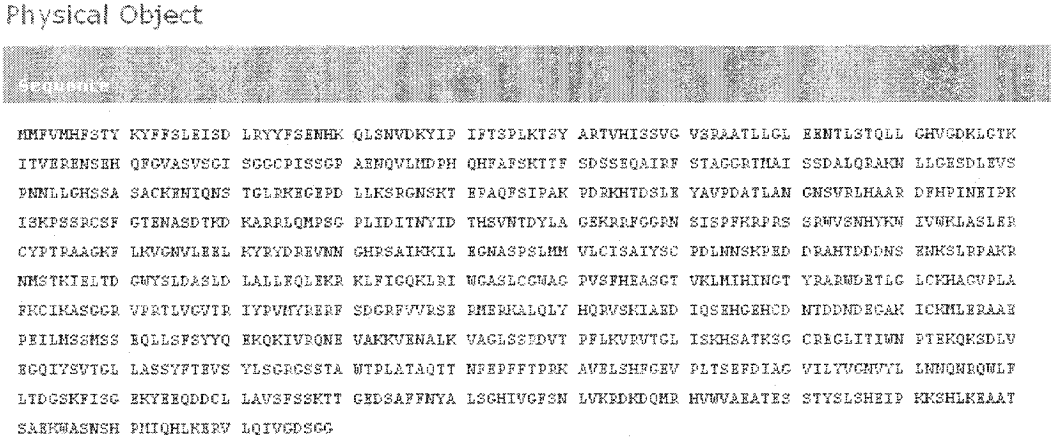
Biozon's web interface comes with a type specific search by which users can search entities of a specific type. This is done by clicking on a type's icon on the Biozon's search panel, Figure 31. Next, the user is given a form in which he/she can set different criteria fields. The criteria fields come from all the sources from which the selected type has been taken. For example, while searching for a protein, the user can specify that the protein should have a specific GI number, which is NCBI's GenInfo number, and also it should be assigned to a specific GO number, which comes from the Gene Ontology.

Criteria fields are different from one entity type to another one, but the fields generally fall into four categories: identifiers (accession numbers, gene names, etc), descriptors (keywords, GO terms, etc), physical properties (length, or resolution), and taxonomy. However, some entity types do not have all these four categories. For example, a UniGene query page contains only an identifiers section and a descriptors section.

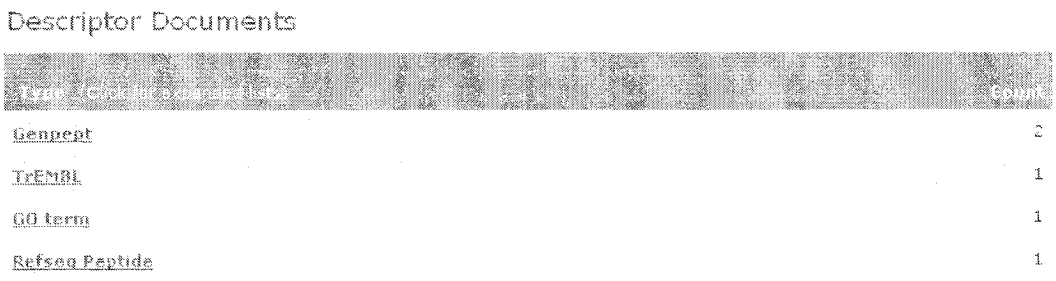
Another search facility in Biozon's web interface is the Quick Search, by which users can keyword search all the entities in the Biozon. This feature is similar to NCBI's Entrez search, but Biozon's quick search also searches on the relations, and at the same time it considers the relation between entities (it returns the results based on the relations that they have with each other) while as described earlier the Entrez is a portal oriented system and does not keep the relation between entities. Thus, Biozon returns more relevant results, and, as Yona et al. [BY06a] claim it is the best



(a) Entity overview, alternate definitions



(b) The physical object



(c) Related descriptor documents

(d) Related objects

Figure 30: Biozon's sample profile page

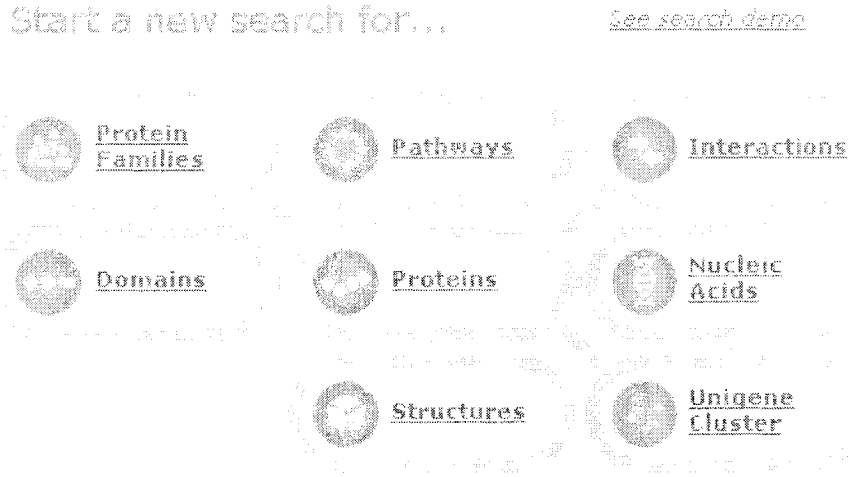


Figure 31: Biozon search panel

quick search tool available so far.

Complex Searches

Performing multiple-entity cross-database queries is not something new between biological researchers. That is, biologists are accustomed to query different data sources to find relevant information of an entity. However, the basic search feature that all biological databases have does not satisfy this need. Although there are some cross database search tools like SRS [EA92], BioMediator [MHTH01], Columbia [TRM+05], and DiscoveryLink [HKR+00], they use the explicit links that are between the different databases and they do not infer the actual relationship that these different databases' contents have. The only exception is Columbia because it links Swiss-Prot protein sequences and PDB chains using similarity.

However, Biozon uses a different strategy to form cross-database queries, which it calls complex queries. Biozon uses its internal graph to form complex queries. That is, the users, through the web interface, define a query graph in steps. The query graph is comprised of query nodes and query edges. Each query node has a data type and some criteria fields, and they together specify which entities should be selected for each query node. In the same fashion, query edges show how the query nodes relate

to each other. In addition, web interface makes sure that the query graph, which is built step by step, is connected at each step. That is, users can not add query nodes that have no edge to any of the currently added query nodes. Next, Biozon runs the query for each query node, and then it tries to find the isomorphic graph between the union of result graphs and the query graph. At this stage, the relationships defined in the query graph will cause the irrelevant entities to be excluded from the final result.

As an example, a complex query can be "Find all the proteins that are associated with breast cancer, and they have a known 3D structure that has a structure with less than 2 Å precision". This complex query returned 2 proteins in the Biozon's October 2005 database.

Fuzzy Searches

Biological data has not been studied completely, and in some cases there is not enough information available about some biological entities. For example, there might not be enough information about functional use of a certain protein. However, biologists have been successful to predict such cases by studying the similar entities. That is, for example, functional use of a protein might be similar to the functional use of its similar proteins.

As described earlier, Biozon stores and updates similarity information of its source data. These similarity data, which are a part of Biozon's derived data, are used by the Biozon's fuzzy search, by which users can look for a specific, for example, protein and at the same time Biozon will look for the other proteins that are similar to it. The reason behind calling this search fuzzy is that Biozon stores a confidence number (e-value) for its similarity relations; therefore, users can specify their desired confidence score while running fuzzy searches. As an example, a user may want to look for proteins that are similar to a protein with P08195 accession number while the confidence score is 1e-100.

Another good point about fuzzy search is that it can be run instantaneously. Each run of fuzzy search is equal to running the given query plus running BLAST on proteins to find similar proteins to the query answer. However, Biozon does not need to compute them at the run time because it stores those similarity relations. Thus, running such a query will be like any normal query, and it should run decently fast. This becomes more important when the result of the given query is a set of entities. In this case, fuzzy search will return the result set and all other similar entities that are similar to either of the result set's members. Obviously, running BLAST for each of the result set's members would take a lot of time, while Biozon can carry the results out at a reasonable time using its offline similarity data.

In addition to normal fuzzy searches, Biozon is also able to form fuzzy complex searches. That is, in some cases, the available biological data is so partial that complex queries do not return any result. For example, Figure 5 shows a fuzzy complex search on enzyme family with EC number 1.1.1.1 which at least has one protein with known structure and that the protein is involved in a certain interaction. The circles in Figure 32. A show that the normal complex search has no result (there is no intersection between the result circles), but the fuzzy search in FigureřefBiozonFuzzySearch. B returns an answer because the protein result set is containing similar proteins as well. Another way to include more proteins in the result is to tune the e-value parameter of the fuzzy search. The bigger the e-value is the more similar proteins are found; thus, the intersection between the results is bigger.

For the moment, as explained by Yona et al. [BY06a], fuzzy search employs the results of BLAST, expression profile similarity, and structural similarity of proteins. Online users have the ability to choose which one of these data to use in their fuzzy searches. In addition, when the results are ready, an indicator will show whether the result has been included by the fuzzy search or it is the result of the query itself. Moreover, the indicator also shows which similarity method has been used for each

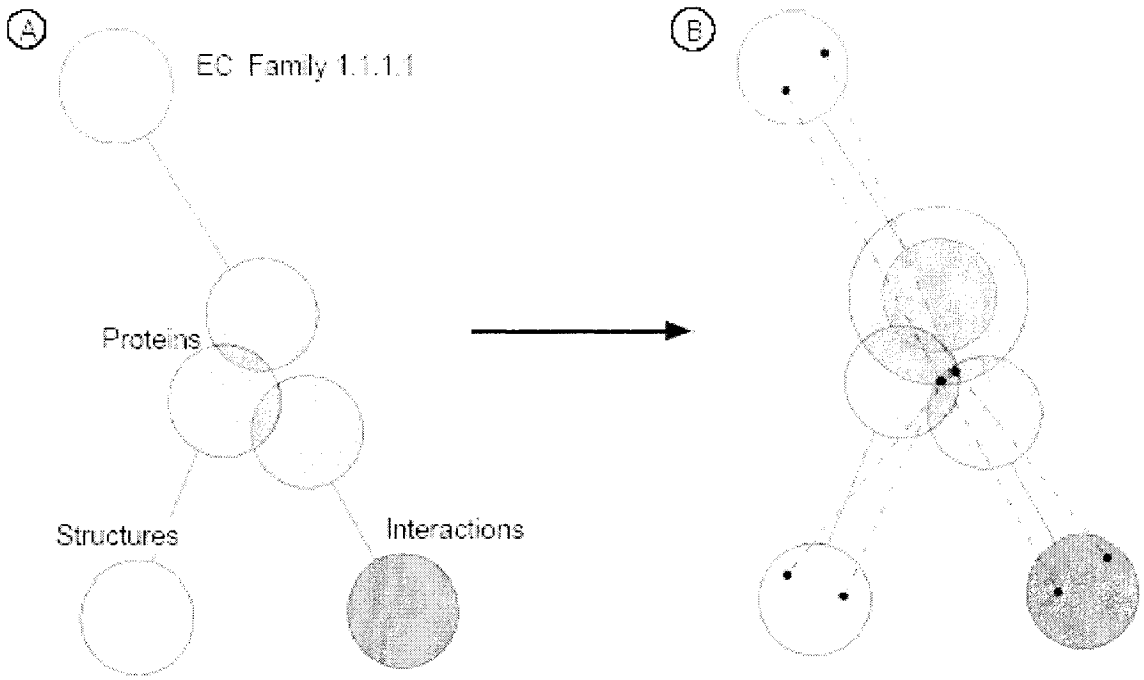


Figure 32: Biozon's fuzzy search

fuzzy result entry.

Ranking

Some databases offer sorting facilities for their results. However, their sorting is mostly based on the techniques that are not useful in the biology context. For example, alphabetical sorting is not always a good way of sorting especially when the query has been run among different datasets. At this stage, there is a need for sorting the results in a way that most proper or relevant entities stand at the top of the results.

Biozon, ranks its results by using PageRank [PBMW98], which is similar to the method that Google uses to rank its results. The way that Biozon uses PageRank is that it looks inside its result graph and tries to find the subgraphs that are most connected to the important data, or the same way it looks for the subgraphs that are most connected. Biozon people call these subgraphs Hubs of Knowledge because they are believed to have the most important knowledge in themselves.

User Account

In the Biozon's web interface, users can open user accounts, where they can save their queries and rerun them later. This is useful when a user needs to rerun a saved query that he/she has done in the past, and he/she wants to have the same results, no matter how many times Biozon's database has been updated since then. In addition to saving and rerunning queries, Biozon users can export and download their query results.

In addition, Biozon has an access privilege system, by which users can be restricted from accessing private datasets. This will give Biozon more flexibility and choice over the kind of datasets it can incorporate.

Analysis tools

Although web interface is the only Biozon's interface, Biozon has some analysis tools which are exposed as services. For the moment Biozon has the following services up and running:

- Users can submit a sequence and get a list of similar sequences.
- Users can submit a sequences and Biozon predicts its domain structure by using Nagarajan et al. [NY04]'s algorithm.
- Biozon has a service that explores different paths in its graph to map human and mouse EST sequences to their protein product.

Some other tools like Meta-DP [SF05] are using Biozon's analysis tools and it is expected that more tools start using it in the future.

Discussion

Biozon stores the data in a complex structure such as a graph. Addition of new data sources and data types needs change of the graph. On the contrary, in the

data warehouse developed in this project, the data is stored in a simple structure, triple of (class, attribute, value). It is effortless to incorporate a new type in the data warehouse, and it only takes defining a new BioXRT input file and loading it into the data warehouse.

In addition, computer programs that access the Biozon graph need to support new types, whereas the computer programs (in the case of this research the agent system) do not need to change when a new type is integrated.

Despite the above, Biozon does not need to update the whole database when a new type is available while the data warehouse loads the whole data whenever an update is available.

However, all the three types of the systems described above (portal oriented, mediator oriented, and data warehouse oriented) need to resolve potential ambiguities in the source data identifiers. In the data warehouse we store the GenBank ID for the genes and proteins because GenBank ID is accepted as a unique id for the genes and proteins.

5.1.4 Taverna

Web services have recently become the main tool for providing programmatic access to biological sources and tools [Ste02], and more organizations are now providing web services. XEMBL [WRR02], openBQS [Sen], Soaplab [SRO03], and DDBJ [MS00] are examples of such web services. As a result of the availability of web services, the need for a workflow system for obtaining data from one service and using it in another service is growing.

Taverna [OAF+04] is an open source workflow system designed for biological researchers to build their own workflows. Taverna workflows are defined as a graph of processors whose output data is the input data to another processor. Taverna is consist of a graphical interface and a Simple Conceptual Unified Flow Language

(Scufl). The graphical interface is where users build their workflows and the Scufl is the language representation of those workflows.

The absence of a standard language to semanticly define workflows led Taverna people to develop their own workflow language, Scufl. Scufl is a high level conceptual XML language that is able to semanticly define the specification of data, processes and resources and put them together to form a workflow. In Scufl, each work is consist of there parts:

- Processors: Processors are the task units. A processor has a name, a list of input data. and a list of output data. During the workflow's execution, each processor can be in either state of initializing, waiting, running, complete, failed, or aborted. Based on the nature of the task, there are different type of processors. The current available types are:
  - Arbitrary WSDL type: This type is the general web service type. Most of the information of a WSDL process is extracted from the web service's WSDL file.
  - Soaplab type: Thie type is a wrapper for the Soaplab services.
  - Talisman type: Talisman processes can invoke a Talisman [Oin03] session. The process need to specify the URL of a Talisman's tscript file in which the service description are defined.
  - Nested workflow type: This type enables invocation of another Scufl workflow. This way, Scufl workflows can act as modules which enables reusability and extensibility.
  - String constant type: This processor type has no input and a single output which is the string constant. This processor can be used as a way of sharing constants between different processors.

- Local processor type: This type is useful when a custom code process is needed in between the workflow. Local processors are Java codes that implement Taverna's interface. Execute SQL Query, XPath from Text, Write Text File, and Send an Email are examples of Taverna's predefined local processors.

Each workflow itself can have a set of input and a set of outputs. Workflow inputs and outputs are annotated by different types of metadata. That is they, they can be associated to a MIME type (which can later be used for rendering purposes), a semantic type from the *myGrid* bioinformatics ontology [WSG+03], and an arbitrary text description.

- Data links: Data links are the connection between processors, and they carry data from one processors to another one. A data link is connected to either the source processor's output data or the workflow's input data, and it is connected to either the destination processor's input or the workflow's output.

- Coordination constraints: Coordination constraint control the execution order or the processors that are not connected with data links. Data links show the dependency of processors and the order they should run; however, some processors do not provide data for each other but they still need to run in a specific order. This processors connect with coordination constraint.

Scufl Workbench

Learning a computer language is not an easy task for biologists; therefore, Taverna comes with a graphical user interface for design and execution of Scufl workflows. The workbench provides enough facilities to design a new workflow from scratch without having any Scufl knowledge.

The Sculf workbench is consist of three main parts. First part is the model explorer in which the workflow is shown in a tree of processors, as shown in Firgure 33.

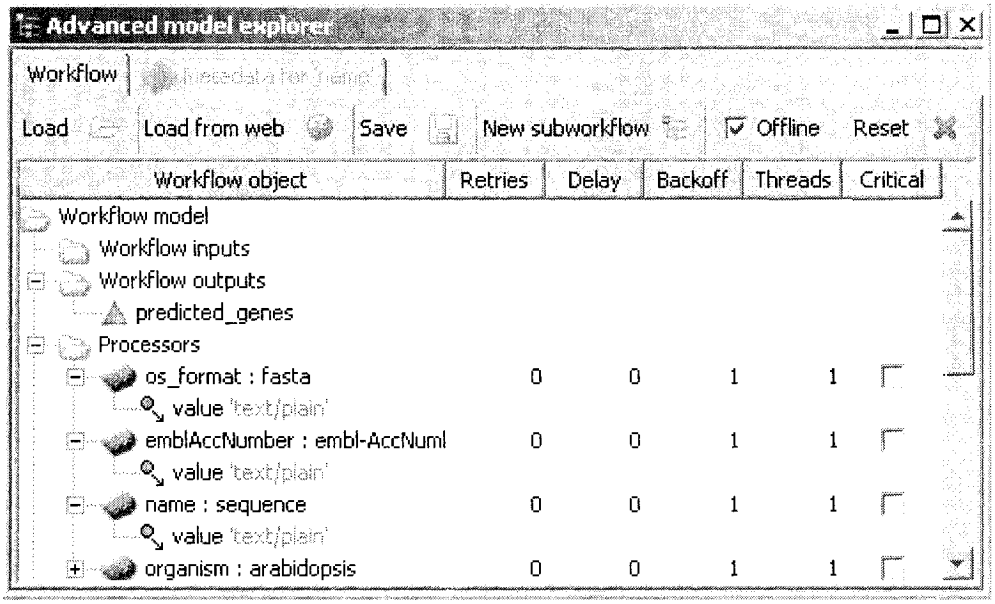


Figure 33: Sculf Workbench – Model Explorer

The next part is the diagram view where a graphical view of the designed workflow is illustrated, Figure 34. The diagram view provide a set of options that help with the illustration of the workflow. For example, it can expand nested workflows. The diagram view draws the workflow by using the Dot tool from GraphViz [GN00], and it is readonly.

The last part is the service explorer, as shown in Figure 35. This is where the available processors are listed, and users can add a processor to the workflow by dragging and dropping processors from this window to the model explorer. The service explorer provides a basic text search facility over the list of the processors.

5.2 Conclusion

Recent enhancements in the technology have made the process of generating new biological data from the existing data sources easier than ever, and it has given it a

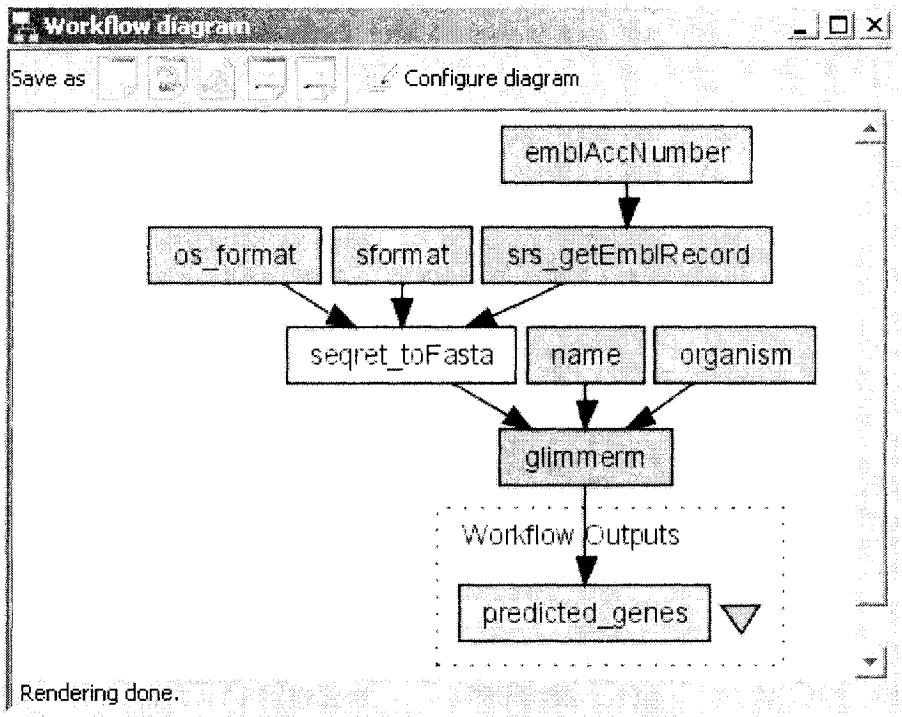


Figure 34: Sculf Workbench – Diagram View

recursive nature. That is, the new data sources are later the basis of other biological data sources. This has changed the life of biologists and bioinformaticians. After all, a daily life of a biologist or a bioinformatician involves discovery of data sources, analyzing them, and moving on to the next source.

In addition, biological sources have a distributed, dynamic, and heterogeneous nature, and finding the right source of data is not a trivial task. Therefore, the need for integrated data sources of biological data grows as the time passes by. In addition to that, many data discovery and analysis tasks done by biologists can be done by computer programs; therefore, computer programs also need to have access to biological data sources and other biological computer programs.

This has been relatively an active field of research for many research teams in the recent years, and many have been successful in developing integrated systems that provide appropriate interfaces for both humans and computer programs. However, this has lead to the vast availability of biological services with no catalog, and it has

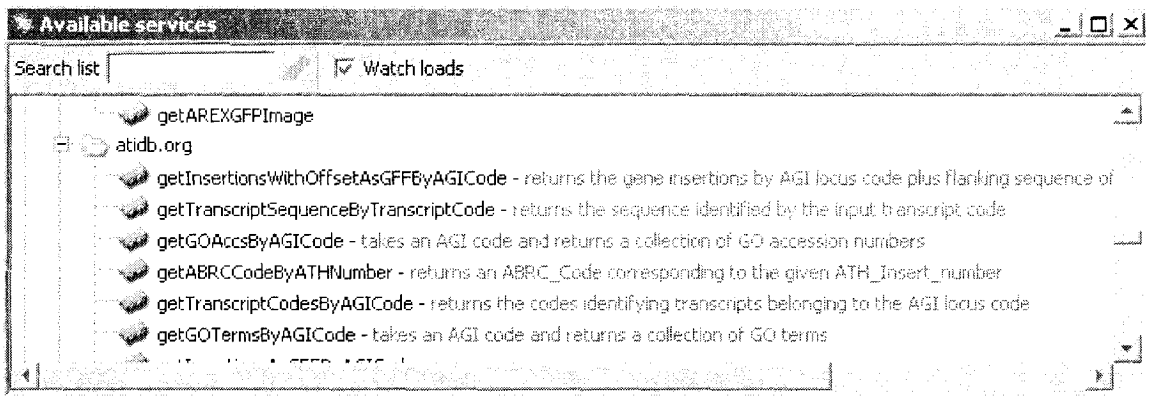


Figure 35: Sculf Workbench – Service Explorer

complicated the process of finding the right service. It is even worst when the diversity of biological vocabularies is taken into consideration. That is, even though there are service catalogs, they are not practical because people use inconsistent vocabularies to refer to the cataloged services.

However, the promise in the field is that a combination of agent systems and the semantic web works fine to solve the problem. In this research we introduce a data warehouse which contains fungal genomics data, and an agent system that maintains the data warehouse.

The data warehouse is the first in its kind to store integrated data from multiple fungal species. In addition, the data warehouse is the first data source on fungal species that stores multiple classification schema (EC, GO, InterPro and FunCat) and the mapping between them. In addition, we also developed an agent based ETL system for the data warehouse systems.

Furthermore, the data warehouse defines a data schema for the integrated data, as described in Appendix B. Moreover, this research involves creation of a list of data sources that contain the input data to the data warehouse, and it involves definition of transformers that change the format of the input data to the design data schema.

The current state of the project is that we have a data warehouse filled with more than 4,000,000 records of data. In addition, the agent system is able to maintain the

data warehouse in terms of integrating new data into it. In more details, the agent system contains the ETL agents and it is ready for the integration of data providing agents.

5.2.1 Evaluation

We have evaluated the performance of the data warehouse by setting up a JMeter test case. JMeter is a Java based tester software developed by Apache people (`http://jakarta.apache.org/jmeter/`). JMeter is able to define a test suite by putting together test components, and it is able to simulate user interaction with the system.

In the case of the data warehouse, we need user interactions with the BioXRT web interface in terms of sending standard HTTP requests and measuring the response time. Figure 36 shows the test case we used to measure the data warehouse response time.

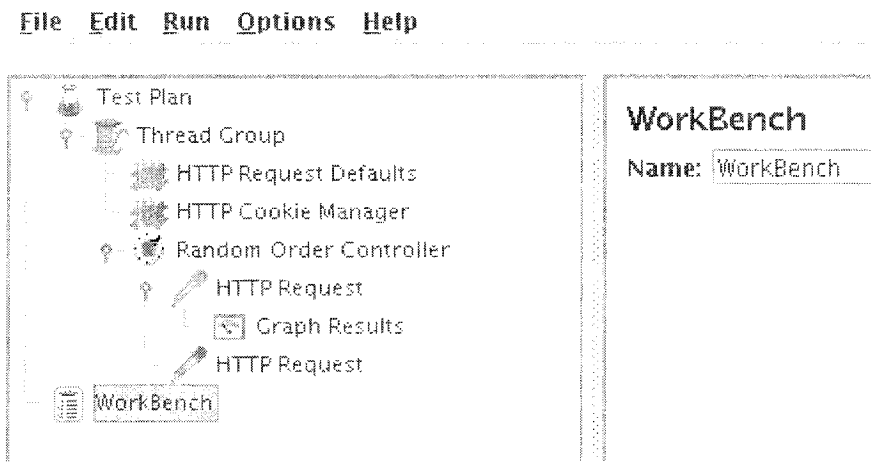


Figure 36: JMeter test case used to evaluate the performance of the data warehouse

The test case contains a `Thread Group` which creates a number of users that concurrently send requests to the BioXRT server and they do not stop until the test is stopped manually. The `HTTP Requests Default` and `HTTP Cookie Manager` node set the default HTTP request behavior (server name and so on) and enable cookies respectively.

The `Random Order Controller` picks one of the HTTP request at a time randomly and runs for the user. This component has been used so that the BioXRT and its database engine will not be biased by the sequence of requests they receive. Under this component there are two HTTP requests to the BioXRT server. One is a query that looks for "All the Neurospora Crassa proteins such that their functions has something to do with Hydrolase" and is measured for performance. The other HTTP request is a random query to make sure BioXRT is not biased.

Users	Average (s)	Median (s)
10	19.3	17
50	83.5	73.9

Table 10: Performance measurement of the data warehouse

Table 10 shows the results of the test case. The test has run on a Pentium 3 Ghz with 1 gigabyte of memory. Furthermore, web server, BioXRT, MySQL database (the database engine to which BioXRT stores the data) and JMeter application all have run on the described machine, and they have not been tuned for performance (default settings only).

The results show BioXRT is able to respond in a reasonable time. Note the results show the performance at a peak time which requires more online users than what is the table because a user needs to read the data before he/she runs another query. In a low traffic situation the data warehouse responds on average within 3 seconds for the measured query.

In terms of scalability our data warehouse system and agent system are fairly scalable. The data warehouse relies on existing relational database engines and web servers which are highly scalable. The agent system also works in a distributed manner and it is always possible to plugin new agents as new power is needed.

We have not done any evaluation of the agent system and usefulness of the data warehouse because a good and fair evaluation requires a team of domain experts and supporting softwares for making inquiries from the system. However, we believe the

data warehouse and the agent system saves biologists' time by giving them a data set for which biologists would need to go to multiple data sources.

In addition, our experience with the system shows use of agents for the development of such systems reduces the effort and the cost. Furthermore, agent systems add to the extendability, the flexibility and robustness of the system as opposed to plain Java objects.

5.2.2 Future Work

Biological data is being generated at a fast pace. Integrating new data as it is made available is a primary goal of developing ETL agents. Therefore, in the future new set of data can be integrated in the data warehouse using the current ETL agents, and new ETL agents may also need to be developed to support more data types.

Furthermore, the ETL agents rely on the developed transformer codes that have the knowledge of data sources. The process of writing new ETL agents involves development of new transformer codes. A nice improvement in this area can be development of template oriented transformers that need a minimal effort to support integration of new data type/sources.

In addition, development of computational agents is a must addition. Computational agents are able to run a remote service and provide the results as a transparent data source. An example would be development of an agent that provides results of a blast request as a data set. Computational agents may also use ETL agents to offload the result of the computations in the data warehouse.

Another nice feature is development of a web service wrapper for the agent system. For the moment, only agent codes that communicate in KQML protocol are able to use the agent system. However, a web service wrapper makes the agent system visible to a wider range of users.

It is also a vision of this research to represent the data in the data warehouse as the

instances of the Fungalweb ontology. However, RACER requires instances in its own storage which includes the concepts (the TBoxes) and the instances (the ABoxes). Therefore, to apply ontology-based access to the data we are required to use RACER and to load the instances into RACER from the data warehouse. In this case there needs to be the appropriate code to read the data warehouse and generate an OWL file that connects the data warehouse to the concepts in the FungalWeb ontology. This takes the definition of the mapping between the data warehouse schema to the concepts in the FungalWeb ontology. We discuss that this mapping is possible because the loaded data in the data warehouse is related to the concepts in the FungalWeb ontology. That is, we use the FungalWeb ontology as the data ontology in the agent system. Therefore, concepts of the FungalWeb ontology are used to tell ETL agent what exact data should be load into the data warehouse.

Finally, development of enhanced update procedures is a nice addition to the data warehouse and the agent system. A complete data load phase may take more than a day. An incremental update policy can be employed to avoid this delay. One candidate for such update policies is development of partial data loads in the BioXRT database. Another feasible policy is to divide the datasets to two sets of more-frequently-changed and less-frequently-changed sets and update only the more-frequently-changed set.

Bibliography

[AAB+01] R. Apweiler, TK. Attwood, A. Bairoch, A. Bateman, E. Birney, M. Biswas, P. Bucher, L. Cerutti, F. Corpet, MD. Croning, R. Durbin, L. Falquet, W. Fleischmann, J. Gouzy, H. Hermjakob, N. Hulo, I. Jonassen, D. Kahn, A. Kanapin, Y. Karavidopoulou, R. Lopez, B. Marx, N.J. Mulder, T. Oinn, M. Pagni, F. Servant, C.J. Sigrist, and E.M. Zdobnov. The InterPro database, an integrated documentation resource for protein families, domains and functional sites. *Nucleic Acids Research*, 29(1):37–40, 2001.

[ABB+00] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, G. M. Rubin, and G. Sherlock. Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nat Genet*, 25(1):25–29, May 2000.

[AS05] A. Al-Shaban. Using semantic web technologies for matchmaking software agents representing web service description. Master's thesis, Concordia University, 2005.

[ASH05] A. Al-Shaban and V. Haarslev. Applying Semantic Web Technologies to Matchmaking and Web Service Descriptions. In *The Proceedings of The Montreal Conference on eTechnologies 2005 (MCeTech2005)*, pages 97–104, Montreal, Canada, January 2005.

[Att02] T. K. Attwood. The PRINTS database: a resource for identification of protein families. *Briefings in Bioinformatics*, 3(3):252–263, 2002.

[AVB01] F. Achard, G. Vaysseix, and E. Barillot. XML, bioinformatics and data integration. *Bioinformatics*, 17(2):115–125, February 2001.

[Bai00] Amos Bairoch. The ENZYME database in 2000. *Nucleic Acids Research*, 28(1):304–305, 2000.

[BAW+05] A. Bairoch, R. Apweiler, C. H. Wu, W. C. Barker, B. Boeckmann, S. Ferro, E. Gasteiger, H. Huang, R. Lopez, M. Magrane, M. J. Martin, D. A. Natale, C. O'Donovan, N. Redaschi, and L. L. Yeh. The Universal Protein Resource (UniProt). *Nucleic Acids Research*, 33:D154, 2005.

[BBB+98] P. G. Baker, A. Brass, S. Bechhofer, C. Goble, N. Paton, and R. Stevens. TAMBIS–Transparent Access to Multiple Bioinformatics Information Sources. In *Int Conf Intelligent Systems for Molecular Biology*, volume 6, pages 25–34, Montreal, Canada, June 1998.

[BBD+00] A. Bateman, E. Birney, R. Durbin, S. R. Eddy, K. L. Howe, and E. L. Sonnhammer. The Pfam Protein Families Database. *Nucleic Acids Research*, 28:263–266, 2000.

[BBH04] C. J. O. Baker, G. Butler, and V. Haarslev. Ontologies, semantic web and intelligent systems for genomics. 2004.

[BDH+95] P. Buneman, S. B. Davidson, K. Hart, C. Overton, and L. Wong. A data transformation system for biological data sources. In *Proceedings of the Twenty-first International Conference on Very Large Databases*, Zurich, Switzerland, 1995. VLDB Endowment, Saratoga, Calif.

[BDW+01] G. D. Bader, I. Donaldson, C. Wolting, B. F. Ouellette, T. Pawson, and C. W. Hogue. BIND — The Biomolecular Interaction Network Database. *Nucleic Acids Research*, 29(1):242–245, January 2001.

[BMC03] S. Bechhofer, R. Möller, and P. Crowther. The DIG Description Logic Interface. In *Proceedings of the International Workshop on Description Logics (DL–2003)*, Rome, Italy, September 2003.

[BSBH06] C. J. O. Baker, X. Su, G. Butler, and V. Haarslev. Ontoligent Interactive Query Tool. In *Proceedings of the Canadian Semantic Web Working Symposium*, volume 2 of *Semantic Web and Beyond: Computing for Human Experience*, pages 155–169, Quebec City, Quebec, Canada, June 2006. Springer Verlag.

[Bur] A. Burger. Agent Technologies in the Life Sciences.

[BWF+00] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The Protein Data Bank. *Nucleic Acids Research*, 28(1):235–242, January 2000.

[BWSN+05] C. J. O. Baker, R. Witte, A. Shaban-Nejad, G. Butler, and V. Haarslev. The FungalWeb Ontology: Application Scenarios. In *Eighth Annual Bio-Ontologies Meeting*, pages 1–2, Detroit, Michigan, USA, June 24 2005.

[BY06a] A. Birkland and G. Yona. Biozon: a hub of heterogeneous biological data. *Nucleic Acids Research*, 34:D235–D242, 2006.

[BY06b] A. Birkland and G. Yona. Biozon: a system for unification, management and analysis of heterogeneous biological data. *BMC Bioinformatics*, 7, 2006.

[CSGK00] F. Corpet, F. Servant, J. Gouzy, and D. Kahn. ProDom and ProDom-CG: tools for protein domain analysis and whole genome comparisons. *Nucleic Acids Research*, 28(1):267–269, 2000.

[DCB+01] S. B. Davidson, J. Crabtree, B. P. Brunk, J. Schug, V. Tannen, G. C. Overton, and C. J. Stoeckert Jr. K2/Kleisli and GUS: experiments in integrated access to genomic data sources. *IBM Systems Journal*, 40(2):512–531, 2001.

[dfG] Wikipedia definition for Glycosylation. `http://en.wikipedia.org/wiki/Glycosylation`.

[DKS+02] K. Decker, S. Khan, C. Schmidt, G. Situ, R. Makkena, and D. Michaud. BioMAS: A Multi-Agent System for Genomic Annotation. *International Journal of Cooperative Information Systems*, 11(3):265–292, 2002.

[EA92] T. Etzold and P. Argos. SRS – an indexing and retrieval tools for flat file data libraries. *Computer Applications in the Biosciences*, 9(1):49–57, 1992.

[EBK+05] J. T. Eppig, C. J. Bult, J. A. Kadin, J. E. Richardson, J. A. Blake, and the members of the Mouse Genome Database Group. The Mouse Genome Database (MGD): from genes to mice — a community resource for mouse biology. *Nucleic Acids Research*, 33:D471–D475, 2005.

[eFe] Accessible at: `http://www.ncbi.nlm.nih.gov/entrez/query/static/eutils_help.html`.

[EHN94] K. Erol, J. Hendler, and D. S. Nau. Semantics for hierarchical task-network planning. Technical report, University of Maryland at College Park, College Park, MD, USA, 1994.

[Fuc94] R. Fuchs. Predicting protein function: a versatile tool for the Apple Macintosh. *Computer Applications in the Biosciences*, 10(2):171–178, 1994.

[GDM03] J. R. Graham, K. S. Decker, and M. Mersic. DECAF - A Flexible Multi Agent System Architecture. *Autonomous Agents and Multi-Agent Systems*, 7(1-2):7–27, 2003.

[GN00] E. R. Gansner and S. C. North. An open graph visualization system and its applications to software engineering. *Software — Practice and Experience*, 30(11):1203–1233, 2000.

[HBB+06] N. Hulo, A. Bairoch, V. Bulliard, L. Cerutti, E. De Castro, P. S. Langendijk-Genevaux, M. Pagni, and C. J. Sigrist. The PROSITE database. *Nucleic Acids Research*, 34(Database issue), January 2006.

[HBC+] E. L. Hong, R. Balakrishnan, K. R. Christie, M. C. Costanzo, S. S. Dwight, S. R. Engel, D. G. Fisk, J. E. Hirschman, M. S. Livstone, R. Nash, J. Park, R. Oughtred, M. Skrzypek, B. Starr, C. L. Theesfeld, R. Andrada, G. Binkley, Q. Dong, C. Lane, B. Hitz, S. Miyasato, M. Schroeder, A. Sethuraman, S. Weng, K. Dolinski, D. Botstein, and J. M. Cherry. Saccharomyces Genome Database. Accesible at: `http://www.yeastgenome.org/`.

[HHH+03] F. V. Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL web ontology language reference, 2003.

[HKR+00] L. M. Haas, P. Kodali, J. E. Rice, P. M. Schwarz, and W. C. Swope. Integrating life sciences data-with a little garlic. In *BIBE '00: Proceedings of the 1st IEEE International Symposium on Bioinformatics and Biomedical Engineering*, page 5, Washington, DC, USA, 2000. IEEE Computer Society.

[HLS04] V. Haarslev, Y. Lu, and N. Shiri. OntoXpl - Intelligent Exploration of OWL Ontologies. In *Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2004)*, pages 624–627, Beijing, China, September 2004.

[HM01] V. Haarslev and R. Möller. Description of the RACER System and its Applications. In *Description Logics*, 2001.

[HM03] V. Haarslev and R. Möller. Racer: A core inference engine for the semantic web. In *Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003)*, pages 27–36, Sanibel Island, Florida, USA, oct 2003.

[HMW04] V. Haarslev, R. Möller, and M. Wessel. Querying the Semantic Web with Racer + nRQL. In *Proceedings of the KI-2004 International Workshop on Applications of Description Logics (ADL'04)*, Ulm, Germany, September 2004.

[HTHG94] D. Higgins, J. D. Thompson, D. G. Higgins, and T. J. Gibson. CLUSTALW: improving the sensitivity of progressive multiple sequence alignment through sequence weighting.position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22:4673–4680, 1994.

[KEG99] KEGG: Kyoto Encyclopedia of Genes and Genomes. *Nucleic Acids Research*, 27(1):29–34, 1999.

[McG01] F. McGeary. *DECAF Programming: An Introduction.* University of Delaware, Department of Computer and Information Science, April 2001.

[MCH03] R. Möller, R. Cornet, and V. Haarslev. Graphical interfaces for Racer: querying DAML+OIL and RDF documents. In *Proceedings of the International Workshop on Description Logics (DL-2003)*, Rome, Italy, September 2003.

[MFHS02] D. L. McGuinness, R. Fikes, J. Hendler, and L. A. Stein. DAML+OIL: An Ontology Language for the Semantic Web. 17(5):72–80, 2002.

[MHTH01] P. Mork, A. Halevy, and P. Tarczy-Hornoch. A model for data integration systems of biomedical data applied to online genetic databases. In *AMIA Annual Symposium*, pages 473–477, November 2001.

[MM04] S. McGinnis and T. L. Madden. BLAST: at the core of a powerful and diverse set of sequence analysis tools. *Nucleic Acids Research*, 32(Web-Server-Issue):20–25, 2004.

[MS00] S. Miyazaki and H. Sugawara. Development of DDBJ-XML and Its Application to a Database of cDNA. *Genome Informatics*, pages 380–381, 2000.

[Neu] Neurospora Crassa Database. Accesible at `http://www.broad.mit.edu/annotation/fungi/neurospora/`.

[NM01] N. F. Noy and D. L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical report, Stanford Knowledge Systems, Laboratory Technical Report KSL-01-05, Stanford Medical Informatics Technical Report SMI-2001-0880, March 2001.

[NY04] N. Nagarajan and G. Yona. Automatic prediction of protein domains from sequence information using a hybrid learning system. *Bioinformatics*, 20(9):1335–1360, 2004.

[OAF+04] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, Ke. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[OBO] Accessible at `http://obo.sourceforge.net/main.html`.

[Oin03] T. Oinn. Talisman-rapid application development for the grid. *Bioinformatics*, 19(1):212–214, June 2003.

[PBMW98] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[PNA+03] S. Peri, J. D. Navarro, R. Amanchy, T. Z. Kristiansen, C. K. Jonnalagadda, V. Surendranath, V. Niranjan, B. Muthusamy, T. K. Gandhi,

M. Gronborg, N. Ibarrola, N. Deshpande, K. Shanker, H. N. Shivashankar, B. P. Rashmi, M. A. Ramya, Z. Zhao, K. N. Chandrika, N. Padma, H. C. Harsha, A. J. Yatish, M. P. Kavitha, M. Menezes, D. R. Choudhury, S. Suresh, N. Ghosh, R. Saravana, S. Chandran, S. Krishna, M. Joy, S. K. Anand, V. Madavan, A. Joseph, G. W. Wong, W. P. Schiemann, S. N. Constantinescu, L. Huang, R. Khosravi-Far, H. Steen, M. Tewari, S. Ghaffari, G. C. Blobe, C. V. Dang, J. G. Garcia, J. Pevsner, O. N. Jensen, P. Roepstorff, K. S. Deshpande, A. M. Chinnaiyan, A. Hamosh, A. Chakravarti, and A. Pandey. Development of human protein reference database as an initial platform for approaching systems biology in humans. *Genome Res*, 13(10):2363–2371, October 2003.

[PPFB03] I. Q. H. Phan, S. Pilbout, W. Fleischmann, and A. Bairoch. NEWT, a new taxonomy portal. *Nucleic Acids Research*, 31(13):3822–3823, 2003.

[PSK+05] S. Pillai, V. Silventoinen, K. Kallio, M. Senger, S. Sobhany, J. Tate, S. Velankar, A. Golovin, K. Henrick, P. Rice, P. Stoehr, and R. Lopez. SOAP-based services provided by the European Bioinformatics Institute. *Nucleic Acids Research*, 33(1), July 2005.

[PSS93] P. F. Patel-Schneider and B. Swartout. Description Logic Specification from the KRSS Effort. Working version (draft), 1993.

[RBG+97] A. L. Rector, S. Bechhofer, C. Goble, I. Horrocks, W. A. Nowlan, and W. D. Solomon. The GRAIL Concept Modelling Language for Medical Terminology. *Artificial Intelligence in Medicine*, 9:139–171, 1997.

[RZM+04] A. Ruepp, A. Zollner, D. Maier, K. Albermann, J. Hani, M. Mokrejs, I. Tetko, U. Güldener, G. Mannhaupt, M. Münsterkötter, and H. W. Mewes. The FunCat, a functional annotation scheme for systematic classification of proteins from whole genomes. *Nucleic Acids Research*, 32(18):5539–5545, 2004.

[SB06] A. Shaneh and G. Butler. Bayesian Learning for Feed-Forward Neural Network with Application to Proteomic Data: The Glycosylation Sites Detection of the Epidermal Growth Factor-Like Proteins Associated with Cancer as a Case Study. In *Advances in Artificial Intelligence*, volume 4013 of *Lecture Notes in Artificial Intelligence*, pages 110–121. Springer Verlag, 2006.

[SCE+04] I. Schomburg, A. Chang, C. Ebeling, M. Gremse, C. Heldt, G. Huhn, and D. Schomburg. BRENDA, the enzyme database: updates and major new developments. *Nucleic Acids Research*, 32(Database-Issue):431–433, 2004.

[Sen] M. Senger. Bibliographic query service.

[SEOK96] G. D. Schuler, J. Epstein, H. Ohkawa, and J. A. Kans. Entrez: Molecular Biology Database and Retrieval System. *Methods in Enzymology*, 266:141–161, 1996.

[SF05] HK. Saini and D. Fischer. Meta-DP: domain prediction meta-server. *Bioinformatics*, 21(12), 2005.

[SFA99] P. Scordis, D. R. Flower, and T. K. Attwood. FingerPRINTScan: intelligent searching of the PRINTS motif database. *Bioinformatics*, 15(10):799–806, 1999.

[SKSB00] C. Schönbach, P. Kowalski-Saunders, and V. Brusic. Data warehousing in molecular biology. *Briefings in Bioinformatics*, 1(2):190–198, 2000.

[SN05] A. Shaban-Nejad. Design and Development of an Integrated Formal Ontology for Fungal Genomics. Master's thesis, Concordia University, March 2005.

[SNBHB05] A. Shaban-Nejad, C. J. O. Baker, V. Haarslev, and G. Butler. The FungalWeb Ontology: Semantic Web Challenges in Bioinformatics and Genomics. In *Semantic Web Challenge - Proceedings of the 4th International Semantic Web Conference*, volume 3729 of *LNCS*, pages 1063–1066, Galway, Ireland, November 2005. Springer-Verlag.

[SRO03] M. Senger, P. Rice, and T. Oinn. Soaplab - a unified Sesame door to analysis tools. In S. J. Cox, editor, *UK e-Science All Hands Meeting*, pages 509–513, September 2003.

[Ste02] L. Stein. Creating a bioinformatics nation. *Nature*, 417:119–120, 2002.

[TRM+05] S. Tril, K. Rother, H. Muller, T. Steinke, I. Koch, R. Preissner, C. Frommel, and U. Leser. Columbia: an integrated database of proteins, structures, and annotations. *BMC bioinformatics*, 2005.

[WBB+06] D. L. Wheeler, T. Barrett, D. A. Benson, S. H. Bryant, K. Canese, V. Chetvernin, D. M. Church, M. DiCuccio, R. Edgar, S. Federhen, L. Y. Geer, W. Helmberg, Y. Kapustin, D. L. Kenton, O. Khovayko, D. J. Lipman, T. L. Madden, D. R. Maglott, J. Ostell, K. D. Pruitt, G. D. Schuler, L. M. Schriml, E. Sequeira, S. T. Sherry, K. Sirotkin, A. Souvorov, G. Starchenko, T. O. Suzek, R. Tatusov, T. A. Tatusova, L. Wagner, and E. Yaschenko. Database resources of the National Center for Biotechnology Information. *Nucleic Acids Research*, 34(Database issue), January 2006.

[Wie92] G. Wienderhold. Mediators in the architecture of future information systems. *IEEE computer*, 21(3):38–50, March 1992.

[WL02] M. D. Wilkinson and M. Links. BioMOBY: an open source biological web services proposal. *Briefings in Bioinformatics*, 3(4):331–341, December 2002.

[Won95] L. Wong. The collection programming language reference manual. Accessible at `ftp://ftp.cis.upenn.edu/pub/papers/limsoon/cpl-defn.ps.gz` (last accessed on June 26, 2006), October 1995.

[WRR02] L. Wang, J. J. Riethoven, and A. Robinson. XEMBL: Distributing EMBL data in XML format. *Bioinformatics*, 18:1147–1148, 2002.

[WSG+03] C. J. Wroe, R. D. Stevens, C. A. Goble, A. Roberts, and M. Greenwood. A suite of DAML+OIL ontologies to describe bioinformatics web services and data. *International Journal of Cooperative Information Systems. special issue on Bioinformatics and Biological Data Management*, 12(2):197–224, July 2003.

[YENS05] B. Y., J. D. Eckart, E. K. Nordberg, and B. W. S. Sobral. ToolBus - An Interoperable Environment for Biological Researchers. In *METMBS*, pages 274–280, 2005.

[ZDKS] J. Zhang, G. E. Duggan, R. Khaja, and S. W. Scherer. Generalized biological database platform based on cross-referenced tables (XRT). Accessible at `http://projects.tcag.ca/bioxrt/` (last accessed on July 3, 2006).

[ZLAE00] E. M. Zdobnov, R. Lopez, R. Apweiler, and T. Etzold. The EBI SRS Server: Recent Developments. In *German Conference on Bioinformatics*, pages 139–148, 2000.

Appendix A

Configurations

A.1 BioXRT

Search Screen

```
#### configuration file for pre-defined tables for BioXRT DB
xrt_db    = fungalweb
host      = localhost
port      =
user      =xrt
pass      =123

description = Concordia FungalWeb Database

page header  = <h>Concordia FungalWeb Database</h2>

#bgcolor      = #E5F5F5
width        = 780
rows         = 20
rows2choose  = 10 20 50 100 200 0
layout       = hHtml
default table =
show back    = 1
link_target  =

### below are the table definitions
#   table key in the square brackets

```

```
[organisms]
view_id       = V0001
title         = Organisms
main_class    = organisms
keyword min len = 0
keyword examples =
filter examples  =
C1.           = 0::LongName::Name
C2.           = 0::Order::Order
C3.           = 0::Phylum::Phylum
C4.           = 0::Kingdom::Kingdom

[org_feature]
view_id       = V0002
title         = Organisms' Features
main_class    = org_feature
keyword min len = 0
keyword examples =
filter examples  =
C1.           = 0::start::Start
C2.           = 0::end::End
C3.           = 0::type::Type
C4.           = 0::comment::Comment
C5.           = 0::source::Source

[gene]
view_id       = V0003
title         = Genes
main_class    = gene
keyword min len = 0
keyword examples =
filter examples  =
C1.           = 0::TaxID::Taxanomy ID
C2.           = 0::GeneID::Gene ID
C3.           = 0::name::Name
C4.           = 0::sequence::Sequence
C5.           = 0::source::Source

[protein]
view_id       = V0004
title         = Proteins
main_class    = protein
keyword min len = 0
keyword examples =
filter examples  =
```

```
C1.            = 0::TaxID::Taxanomy ID
C2.            = 0::ProteinID::Protein ID
C3.            = 0::name::Name
C4.            = 0::sequence::Sequence
C5.            = 0::source::Source

[go_hierarchy]
view_id        = V0005
title          = GO Hierarchy Nodes
main_class     = go_hierarchy
keyword min len = 0
keyword examples =
filter examples  =
C1.            = 0::parent::GO Tree Node
C2.            = 0::descendant::Descendant

[go_nodes]
view_id        = V0006
title          = GO Nodes
main_class     = go_nodes
keyword min len = 0
keyword examples =
filter examples  =
C1.            = 0::GO::GO Number
C2.            = 0::name::Name
C3.            = 0::name_space::Name Space
C4.            = 0::definition::Definition
C5.            = 0::alternative_ids::Alternative IDs

[funcat_hierarchy]
view_id        = V0007
title          = Funcat Hierarchy Nodes
main_class     = funcat_hierarchy
keyword min len = 0
keyword examples =
filter examples  =
C1.            = 0::node::Funcat Tree Node
C2.            = 0::descendant::Descendant
C3.            = 0::level::Level

[funcat_nodes]
view_id        = V0008
title          = Funcat Nodes
main_class     = funcat_nodes
keyword min len = 0
```

```
keyword examples =
filter examples  =
C1.            = 0::node::Funcat Node
C2.            = 0::description::Description

[funcat]
view_id        = V0009
title          = Funcat Information
main_class     = funcat
keyword min len = 0
keyword examples =
filter examples  =
C1.            = 0::TaxID::Taxanomy ID
C2.            = 0::ProteinID::Protein ID
C3.            = 0::funcat::Funcat Number
C4.            = 0::organism_name::Organism Name
C5.            = 0::protein_name::Protein Name
C6.            = 0::sequence::Sequence
C7.            = 0::source::Source

[mips2go]
view_id        = V0010
title          = MIPS to GO mapping
main_class     = mips2go
keyword min len = 0
keyword examples =
filter examples  =
C1.            = 0::FuncatID::Funcat ID
C2.            = 0::GoID::GO ID
C3.            = 0::funcat_description::Funcat Description
C4.            = 0::go_description::GO Description

[ec_hierarchy]
view_id        = V0011
title          = EC Hierarchy Nodes
main_class     = ec_hierarchy
keyword min len = 0
keyword examples =
filter examples  =
C1.            = 0::parent::EC Tree Node
C2.            = 0::descendant::Descendant
C3.            = 0::level::Level

[ec_nodes]
view_id        = V0012
```

```
title          = EC Nodes
main_class     = ec_nodes
keyword min len = 0
keyword examples =
filter examples  =
C1.            = 0::EC::EC Number
C2.            = 0::definition::Description

[enzyme_pathway]
view_id        = V0013
title          = ECs in Pathways
main_class     = enzyme_pathway
keyword min len = 0
keyword examples =
filter examples  =
C1.            = 0::EC::EC Number
C2.            = 0::pathway_number::Pathway Number
C3.            = 0::pathway_title::Pathway Title
C4.            = 0::pathway_category::Pathway Category
C5.            = 0::pathway_link::Pathway Link

[ec2go]
view_id        = V0014
title          = EC to GO mapping
main_class     = ec2go
keyword min len = 0
keyword examples =
filter examples  =
C1.            = 0::EcID::EC Number
C2.            = 0::GoID::GO Number
C3.            = 0::go_description::GO Term Description

[interpro_scheme]
view_id        = V0015
title          = InterPro Scheme
main_class     = interpro_scheme
keyword min len = 0
keyword examples =
filter examples  =
C1.            = 0::InterProID::InterPro ID
C2.            = 0::short_name::Short Name
C3.            = 0::name::Name
C4.            = 0::type::Type

[interpro2go]
```

```
view_id       = V0016
title         = InterPro to GO mapping
main_class    = interpro2go
keyword min len = 0
keyword examples =
filter examples  =
C1.           = 0::InterProID::InterPro ID
C2.           = 0::GoID::GO ID
C3.           = 0::interpro_description::InterPro Description
C4.           = 0::go_description::GO Description
```

Appendix B

Data Files

This appendix explains the structure of the BioXRT data files.

Column	Description
ID	BioXRT ID
EcID	Enzyme Category ID
GoID	Gene Ontology ID
go_description	Corresponding Gene Ontology description

Table 11: ec2go.xrt

Column	Description
ID	BioXRT ID
parent	Parent EC node
descendant	Descendent node in the tree
level	Depth of the descendant node

Table 12: ec_hierarchy.xrt

Column	Description
ID	BioXRT ID
EC	EC node ID
definition	EC node definition

Table 13: ec_nodes.xrt

Column	Description
ID	BioXRT ID
EC	EC ID
pathway_number	Pathway ID
pathway_title	Pathway title
pathway_category	Pathway category
pathway_link	The link to the kegg's pathway profile page

Table 14: enzyme_pathway.xrt

Column	Description
ID	BioXRT ID
TaxID	Taxonomy ID of the organism
ProteinID	Protein for which this function has been assigned
funcat	Functional category of this assignment
organism_name	organism name
protein_name	protein name
sequence	protein sequence
source	The source from which this assignment has been taken

Table 15: funcat.xrt

Column	Description
ID	BioXRT ID
node	FunCat node ID
descendant	Descendant FunCat node
level	Depth of the descendant node

Table 16: funcat_hierarchy.xrt

Column	Description
ID	BioXRT ID
node	FunCat node ID
description	The corresponding definition of the node

Table 17: funcat_nodes.xrt

Column	Description
ID	BioXRT ID
TaxID	Taxonomy ID of the organism
GeneID	GenBank ID of the gene
OtherIDs	Other databases' ID for this gene
name	Gene name. It's been either taken from the original source or from the GenBank unless the original source has the name
sequence	Gene DNA sequence
comment	Other data taken from GenBank and source data
source	The name of the source data's institute.

Table 18: genes.xrt

Column	Description
ID	BioXRT ID
TaxID	Taxonomy ID of the organism
ProteinID	GenBank ID of the protein
OtherIDs	Other databases' ID for this protein
name	Protein name. It's been either taken from the original source or from the GenBank unless the original source has the name
sequence	Protein RNA sequence
comment	Other data taken from GenBank and source data
source	The name of the source data's institute.

Table 19: proteins.xrt

Column	Description
ID	BioXRT ID
ProteinID	Protein ID studied in this annotation
TaxID	Taxonomy ID of the organism whose protein is studied
GOID	Gene Ontology assignment
EvidenceSource	Evidence source. For example a UniProt protein
Evidence	Evidence type
Aspect	Gene Ontology aspect: Modelcular Function, and ...

Table 20: go_hierarchy.xrt

Column	Description
ID	BioXRT ID
parent	Gene Ontology term
descendant	Descendant Gene Ontology term

Table 21: go_annotation.xrt

Column	Description
ID	BioXRT ID
GOID	Gene Ontology term
name	Name of the term
name_space	Gene Ontology aspect: Molecular Function, and ...
definition	Definition of the term
alternative_ids	Alternative Gene Ontology terms

Table 22: go_nodes.xrt

Column	Description
ID	BioXRT ID
InterProID	InterPro ID
GoID	Gene Ontology term ID
interpro_description	InterPro node description
go_description	GO term description

Table 23: interpro2go.xrt

Column	Description
ID	BioXRT ID
InterProID	InterPro node ID
short_name	Short name
name	full node name
type	node type: Domain, and ...

Table 24: interpro_scheme.xrt

Column	Description
ID	BioXRT ID
FuncatID	FunCat node ID
GoID	Gene Ontology term
funcat_description	FunCat node description
go_description	Gene Ontology term description

Table 25: mips2go.xrt

Column	Description
ID	BioXRT ID
LongName	Organism's full name
Order	Order
Phylum	Phylum
Kingdom	Kingdom

Table 26: orgasnims.xrt

Column	Description
ID	BioXRT ID
XRTID	The protein file's BioXRT ID
ProteinID	GenBank protein ID
ProteinOtherIDs	Other protein IDs taken from original sources
TaxID	Taxonomy ID for the organism
Start	Start of matching area
End	End of matching area
Score	E score
Status	Status
Evidence	Evidence for the matching
MatchID	Matched protein's ID
MatchName	Matched protein's name
DatabaseName	The database from which the matched protein is taken
InterProID	InterPro ID for matching
InterProName	InterPro name for matching
InterProType	InterPro type of matching

Table 27: interpro_protein.xrt

Column	Description
ID	BioXRT ID
XRTID	The gene file's BioXRT ID
ProteinID	GenBank gene ID
ProteinOtherIDs	Other gene IDs taken from original sources
TaxID	Taxonomy ID for the organism
Start	Start of matching area
End	End of matching area
Score	E score
Status	Status
Evidence	Evidence for the matching
MatchID	Matched gene's ID
MatchName	Matched gene's name
DatabaseName	The database from which the matched gene is taken
InterProID	InterPro ID for matching
InterProName	InterPro name for matching
InterProType	InterPro type of matching

Table 28: interpro_gene.xrt