USER-ORIENTED SEMANTIC SERVICE DISCOVERY

A thesis submitted to the University of Manchester
for the degree of Master of Philosophy
in the Faculty of Science and Engineering

2004

By
Pinar Alper
Department of Computer Science

Contents

List of Tables

List of Figures

Abstract

The need for discovering components has existed since the emergence of networks and distributed computing. Recently, developments on service based distributed computing and the semantic web are beginning to enable flexible service based architectures, where services can be discovered and composed into workflows for achievement of high level tasks. Currently there exist a large amount of research interest focused on providing fully automated service discovery and composition. In this thesis we describe the requirements from the Bioinformatics domain, particularly the myGrid project, in which a semi-automated approach is desired where the users are chiefly in charge of selecting and composing services rather than unattended software agents. We report on the *User Oriented* semantic service discovery system, Feta, that has been developed based on the domain's requirements and outcomes of previous service discovery solutions in myGrid. Our findings point out that a discovery system that is to be deployed in a real life bioinformatics setting, and is expected to assist *users*, should support lightweight semantic descriptions and a user-oriented model of services.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institution of learning.

Copyright

Acknowledgements

I would like to express my deepest gratitude to my supervisor Prof. Carole Goble, who has always encouraged me and guided me during my studies. I would also like to thank Dr. Phillip Lord and Dr. Chris Wroe for their patience with my endless questions and their guidance throughout the project.

Finally special thanks to my husband Koray Alper for his caring, patience and support.

Chapter 1

Introduction

The concept of discovery has been a focal point of attention since the emergence of networks and distributed computing. It is possible to find incarnations of the discovery concept within nearly every distributed computing paradigm. These components have a crucial role in sharing and orchestrated use of diverse types of resources in closed or open environments.

Recently, the emergence and widespread use of web based protocols and service orientation has introduced possibilities for open and flexible service based architectures where services can be discovered and composed into workflows. Moreover, openness of the environment has fuelled research on providing unambigious semantic descriptions of services to cater for increased automation in service discovery and composition activities known as the Semantic Web Services research.

In this thesis we focus on discovery. Initially we take a general approach and describe its role in distributed environments, the motivations behind it, and the common way it is performed. Then we describe the reflections of these as a set of discovery requirements in the bioinformatics domain, and particularly the myGrid project, where the users are chiefly in charge of selecting and composing services rather than unattended software agents. We then analyze stereotypical discovery systems, Traders, in different distributed environments explaining how each adopts an information model and a discovery mechanism to meet the particular environment's needs.

We investigate the role of the Trader Information Model and the discovery mechanisms that is to be adopted in myGrid's environment. We report on the

User Orientated semantic service discovery system, Feta, that has been developed based on the domain's requirements and the outcomes of previous service discovery activities in myGrid. Our findings point out that a discovery system that is to be deployed in real life applications, and is expected to assist users in discovery should support light-weight semantic descriptions, and a user-oriented model of services.

In this chapter we first describe the general set of motivations behind discovery systems in distributed environments. We will then provide an identification of common characteristics of discovery systems. Later the context in which our work falls is described by giving an overview of the myGrid Project and the role of discovery within myGrid. Finally the road map for the thesis is given.

1.1 Motivation for Discovery

Distributed systems such as the Web or the Grid are composed of large numbers of components and large numbers of bindings between these components and their users. Examples of these components can be:

- Compute Resources, which are defined as *"systems accessible via a network"*[27] on the Grid (e.g. clusters and file servers).
- Distributed software objects shared within enterprise application integration platforms (e.g. Common Object Request Broker Architecture CORBA).
- Services, which are defined as *"network-enabled entities that provide some capability through the exchange of messages"* [27] on the Web.

In order to be able to create bindings to either of these components, a mechanism for discovering them is required. The reasons for this are:

- It would not be feasible for requesters to perform discovery on their own given the large number of components in the environment.
- Unanticipated need for (re)use of components may emerge during operation of distributed systems.
- To make distributed systems more reliable static bindings to components should be avoided. Dynamic bindings to components should be established whenever needed.
- Discovery is also needed for component mobility. Users of a component need not be aware of the location of it or its binding details until the component is required.

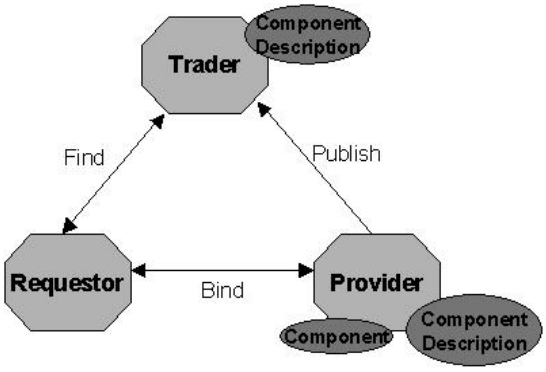


Figure 1.1: Common Discovery Architecture in Distributed Environments.

Based on the above motivations, specialized components in the distributed environment are developed to aid requesters in discovering components to be able to establish a binding to them. These have been defined as middle-agents [28] or traders [70] in the literature.

Middle-agents mediate between providers and requestors of components, and therefore provide a loosely coupled architecture between them. The middle agents may have different behaviours [28]. "Facilitators/Brokers" actively take part during the establishment of the binding between the requestor and provider. Whereas "Matchmakers" also known as "Yellow Page/Directory" or "Trader" middle agents collect adverts of components from providers to aid requestors in their search without intervening in the binding stage. The focus of our attention will be on the latter type of agents that aid the discovery stage only. We will use the term Trader to refer to these types of agents within our analysis.

In the following sections, we will introduce common characteristics of discovery systems briefly. Particular examples of these systems will be given in Chapter 2.

1.2 Common Discovery Architecture

The commonly observed discovery architecture and interactions within this architecture can be seen in Figure 1.1. There exist three parties in the architecture namely the Trader, the Requestor and the Provider. The operations within this architecture are as follows:

- The Provider **publishes** (or registers/advertises) descriptions of its services to the Trader. The advertisements stored within the Trader are *component*

descriptors that describe what the component at the provider's side has to offer together with its binding details. The publication might either be temporary or permanent. In certain distributed environments like the Grid, up to date information on the status of frequently changing/failing components is vital, so temporary publishing, also known as soft-state registration or component description leasing, is needed [27]. In such cases the provider repeatedly performs the publish operation to announce the presence of its components on the network

- Requestors in need of a component contact the Trader to make an **inquiry** about existing components that match their requirements. The trader uses a *matching method* to answer the inquiry and returns information on resulting components to the requestor. Matchmakers are special kinds of Traders that also take the Provider's preferences on potential users of their components into account during matching.
- By making use of the information obtained from the Trader the requestor **binds** to the provider's component to obtain the desired service.

While Figure 1.1 displays a single Trader assisting discovery, multiple Traders may as well work in cooperation to answer discovery requests. Peer to peer organization, or federations of Traders are also common in distributed environments where the discovery function is designed to scale when the number of providers and requestors increase. We should note that the architecture and its operations displayed in Figure 1.1, where Providers and Requestors are de-coupled from each other by the Trader, is not the only existing approach to discovery. Alternatives such as discovery without Traders via multi-cast [90] also exists; however, we will limit our analysis to the Provider-Trader-Requestor architecture and its operations throughout the thesis.

1.2.1 Component Descriptors

Component descriptors within the Trader are composed of names and/or properties. Traders specialized in component **lookup** only provide unique names as component descriptors. Traders with unique naming can employ Hierarchical or Flat naming schemes., whereas in other Traders, which are focused on **discovery** rather than lookup, the descriptor is a group of properties one of which may be the name of the component.

Component properties published to a Trader are descriptions ranging from

simple (e.g. attribute value pairs) to complex (e.g. a conceptual description in an AI-based formalism). The content of descriptions can be composed of, but not limited to, information on:

- Component capabilities like what the component does or what the component is;
- Component provider details;
- Component status, or availability;
- Providers' usage policies on the component;
- Component accessibility details.

Depending on whether the discovery is performed within a closed or open environment the information model supported by the Trader can accordingly be:

1. **Specific**, so that descriptions conform to a commonly agreed information model which predefines what the component capabilities are [10][1]. An example of this type of description could be found within a trader in a grid environment that predefines capabilities of resources by specifying types for them (e.g. a resource can be defined to be either a workstation or a storage device on the network) and also predefining the properties that they may have (e.g. a workstation can be defined to have properties such as CPU load, memory size, etc.)
2. **Generic**, so that the information model defines the minimal commonly agreed characteristics of the shared components without predefining component capabilities. These models need to be augmented with knowledge of different domains in an open environment to be able to make descriptions of capabilities. An example of such a situation is given in Figure 1.2 where the components subject to discovery are Web Services (i.e. software applications with web enabled interfaces). Web services can be used in a variety of domains for providing different functionalities. To be able to give a complete description of the web service its description consists of two parts. The generic part [20] [86] models a service to have endpoints operating over input/output messages of certain data types. The specific part models the functionality of the service (e.g. HotelBooking) and what messages the service consumes (e.g. reservationNo).

Whichever the nature of the information model is, the descriptions are generally mapped to an underlying back-end data model/schema (e.g. eXtensible Markup Language (XML), Resource Description Framework (RDF) or Relational

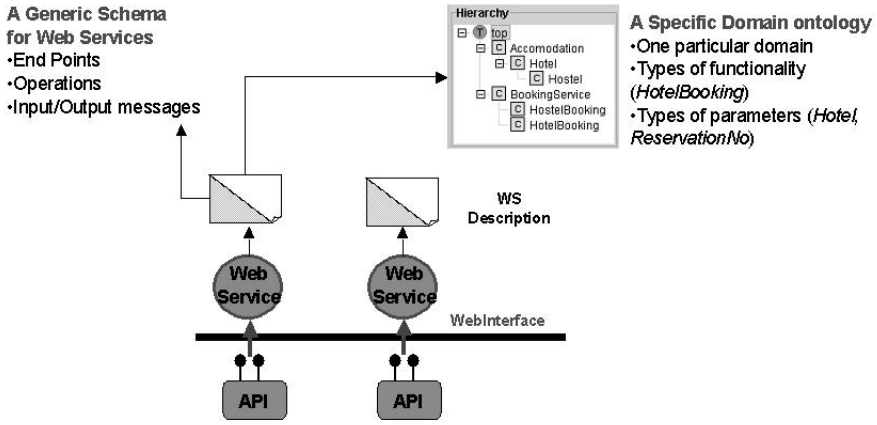


Figure 1.2: An Example of Generic Resource Description Model and its augmentation with Domain Knowledge. The figure is a simplified form of its original in [78].

Schema) for storage and retrieval within the Trader.

1.2.2 Discovery Mechanism

Based on the descriptors stored, Traders provide two basic functionalityies for discovery. These are discovery by name (i.e. lookup) and discovery by properties.

Traders providing lookup functionality perform a unique name to binding details mapping for components. Hierarchical unique names inherently support lookup via browsing the name hierarchy.

Traders providing discovery by properties allow requestors to make search requests/queries over the component's properties. The matching mechanism could also range from simple to complex depending on the nature of descriptions. An example simple matching mechanism could answer questions like "find a linux machine with CPU Load less than 20 percent" where the types of resources and their attributes are all pre-defined by the Information Model supported by the Trader. The requestor only supplies the desired resource type and attribute value. A complex matching mechanism would be a logical subsumption checking algorithm that decides whether there exist any offers whose conceptual description subsumes the request.

The matching mechanism of the Trader could be offered through an Application Package Interface (API) that reflects the Trader's information model or it can be offered as solely reliant on the back-end storage schema (e.g. Relational

Schema) and its querying capabilities (e.g. SQL).

Depending on the algorithm used during the matching the inquiry results can be composed of exact matches and inexact matches. The trader might also support ranking of the exact or inexact matches to further assist the selection process.

1.2.3 Purpose of Discovery

Discovery can be done for different purposes, and the purpose of discovery has implications for the descriptions of components.

In cases where the discovered components are information consuming / producing or world-altering entities such as distributed objects or web services, an orchestrated use of them to achieve a higher level task may be desirable. For example a client (a human, or a software agent) may be searching for a credit card balance checking service to use it together with a hotel booking service that accepts credit card balance confirmation and produces a reservation number. Such discovery can be defined as discovery for composition. Descriptions of composable components generally include information on pre-conditions that need to be satisfied prior to establishment of a binding to the component, or post-conditions/effects that will take place after use of the component.

On the other hand in certain environments such as the computational Grid, where discovery is performed for tasks like job submission, orchestrated use of discovered components is rarely required. Therefore discovery is done for the purpose of discovery only.

1.2.4 Stakeholders in Discovery

All of the interactions in the discovery architecture take place between software agents. These agents act either on behalf of human users or themselves:

- The Providers of component descriptions may be either human users or software agents. Examples of human provided descriptions are textual descriptions for components, or categories or name hierarchies that the components can be grouped under. On the other hand an example of a solely machine generated description could be a periodically produced load status report by a computer cluster that would like to announce its presence and free CPU cycles to the network.

- The Requestors for components may also be human-users or software agents. An example of a software agent that initiates discovery could be a process scheduling application that is in need of clusters with free CPU cycles to share, or a more intelligent web agent, or a human-user looking for a hotel reservation service to compose with other services to achieve a high level task like travel itinerary generation.
- The Trader agent within the architecture operates without human intervention with respect to a pre-defined matching procedure. However, there are middle-agents such as negotiators that we have excluded from our analysis which may refer to interactions with human users during discovery.

To this end we have described the common characteristics of discovery systems. Now we will introduce the context in which our discovery system, Feta, has been developed.

1.3 e-Science and myGrid Project Overview

Computation is being increasingly used in real life activities and Science is no exception to this. e-Science [30], which can be defined as collaborative use of diverse computational resources — instruments, databases, applications, network resources — to assist scientific activities, is currently an active area of research and development.

Among all scientific disciplines Biology has been one of the pioneers that have adopted the use of computational resources to undertake its activities. A new class of experiments termed "*in silico*" has been defined by this community. Biological *in silico* experiments complement traditional experiments by using computational analysis methods to process data, which is obtained from traditional wet laboratory bench experiments. Both the biological data residing in several information repositories, and the tools used for processing that data are highly fragmented and autonomous in nature; therefore integrating them is a serious challenge. Until recently *in silico* experiments were conducted in an *ad hoc* fashion. Almost all data and tools were made accessible to the web via web applications and bioinformaticians had to do all the integration with minimal automation support (i.e. copying and pasting data across web forms, writing screen scraping, or data format converting scripts).

myGrid [83] is a UK e-Science project that is being undertaken to address the

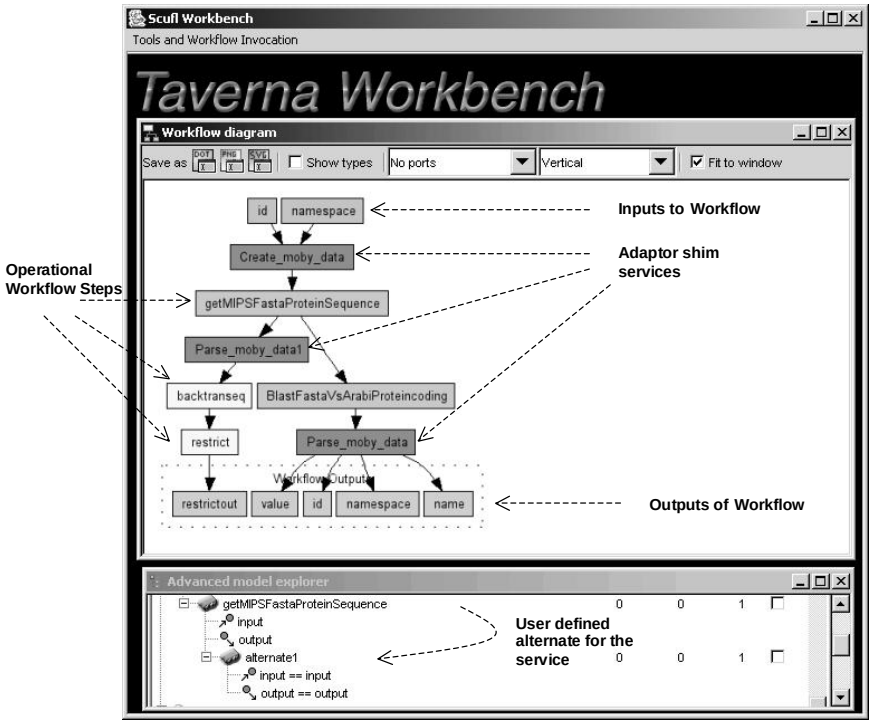


Figure 1.3: An *In Silico* Experiment as a Workflow.

integration problem in biological *in silico* experiments by exploiting the Grid technologies [27]. Grid technologies aim to enable large-scale, and dynamic assembly of diverse resources into transient confederations named Virtual Organizations. Due to the fragmented and autonomous nature of resources (i.e. data and tools) within the bioinformatics domain, myGrid's focus is more on Information Grid, where the challenge is the integrated use of heterogeneous information providing resources rather than the traditional sharing of large-scale volatile compute resources.

Within myGrid *in silico* experiments are formalized either as workflows or distributed queries which integrate information providing resources. An example assembly of bioinformatics resources in an *in silico* experiment (formalized as a workflow) can be seen in Figure 1.3. The figure displays a screenshot of myGrid's workflow development environment Taverna[1][71]. The description for the sample *in silico* experiment in Figure 1.3 is as follows:

1. The scientist starts the experiment by fetching the Arabidopsis Protein Sequence for a certain protein ID from the Munich Information Center for

[1] `http://taverna.sourceforge.net/`

Protein Sequences (MIPS) Database.

2. Later a sequence similarity search for this sequence is done against known Arabidopsis protein coding genes in the MIPS database
3. In parallel to this, the sequence is converted to a format compatible with the EMBOSS[2] toolsuite [77] and back translated using the backtranseq tool of EMBOSS to make a best estimate of the likely nucleic acid sequence it could have come from.
4. Finally the resulting nucleic acid sequence is processed by the restriction tool in EMBOSS toolsuite to find restriction enzyme cleavage sites on the sequence

The screenshot also displays functionally neutral adaptor services within the workflow. These services, called "Shims" [47] are used to overcome syntactic incompatibilities among services and they are functionally neutral within the *in silico* experiment context. The workflow designer can also define alternates for the functionally significant services in the workflow. These alternates are invoked in case their counterparts fail to execute during a workflow run.

The core set of services that myGrid provides are centred around allowing integration of information providing resources (i.e. tools and databases). These are workflow enactment, distributed query processing, and resource discovery services. In addition to these the project aims at providing provenance management, change notification and personalization services to better support the e-Science scientific process [83].

Among the core services of myGrid, resource discovery is the one that is the focus of our work described in this thesis. To be able to position Resource Discovery within to the overall myGrid architecture we want to describe the myGrid *in silico* experiment lifecycle in detail here.

1.4 myGrid *In silico* Experiment Lifecycle

As stated in the previous section, one way of formalizing *in silico* experiments in myGrid is as workflows: it is this workflow based integration approach that we have chosen to provide discovery support for. We see resource discovery is an integral part of workflow design during which the building blocks of the workflow are obtained. Each step in these workflows represents a stage in the *in*

[2] `http://www.hgmp.mrc.ac.uk/Software/EMBOSS/`

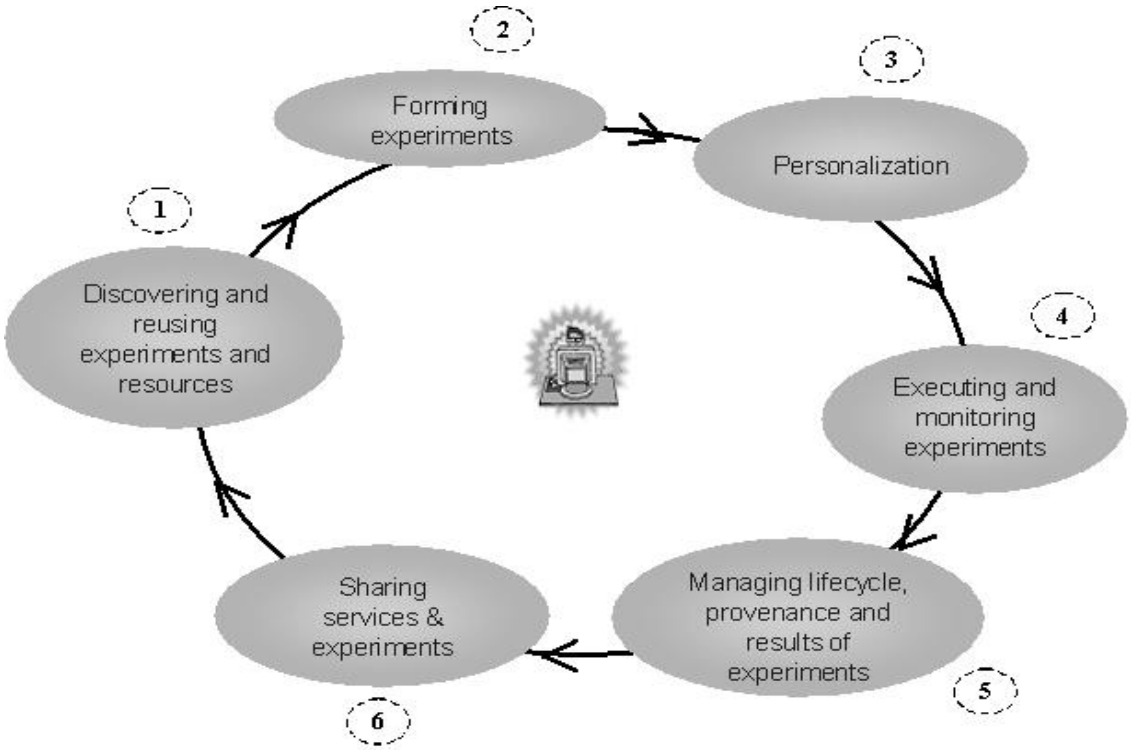


Figure 1.4: *In silico* Experiment Lifecycle in myGrid.

silico experiment during which biological data is processed using bioinformatics services (i.e. bioinformatics resources –applications, tools, databases– exposed as services)[40].

Figure 1.4 depicts the *in silico* experiment life cycle in myGrid. As observed from the figure myGrid components aid biologists to (1) discover services or previously designed workflows (i.e. experiments), (2) edit discovered workflows or build new ones from scratch using the discovered services, (3) personalize services and workflows by attaching usage experience or comments metadata to their descriptions, (4) execute workflows and monitor their executions, (5) manage temporary and final results of executions, (6) and share these workflows with the community by publishing them.

The overall architecture and interactions of myGrid components that aid each activity in the experiment lifecycle can be seen in Figure 1.5. The scientist can interact with the key components of the system through the workflow workbench Taverna.

The discovery stage of the lifecycle (See Figure 1.4 Stage 1) is achieved by the

use of the Discovery Framework (See Figure 1.5 Stage 1). However, use of the discovery framework is not the only mechanism to find and incorporate services into workflows. Users can also discover services by a 'Scavenging' facility provided by the workflow workbench. Scavenging requires users to provide a link to a low-level service descriptions documents or service end points at a particular service provider site. Using this link the workbench harvests all available services hosted at the particular site. Service selection is followed by the experiment design where services are composed into workflows (Stage 2 in both figures), which is the main functionality of the workbench component. During design the user may wish to add personal metadata (Stage 3 in both figures), for example experiences, comments, or thoughts to the set of descriptions of services/workflows in hand that have been retrieved during the discovery stage. This way the discovery process within a research group would improve by the sharing of experience. Once the design is complete it is sent to the enactor component for execution. The workbench interacts with the enactor on behalf of the user during execution and monitoring of workflows (Stage 4 in both figures). The Enactor stores any temporary or final result to the information repository (Stage 5 in both figures) to enable sharing of these results within the community. Finally once the workflow execution is complete the user may publish this workflow with the intention of future re-use (Stage 6). This way the experiment lifecycle is completed to be re-initiated with the discovery stage.

In the service-oriented distributed setting, where myGrid components are deployed and used, one of the most important services is the resource discovery service. The large number of activities that are achieved through the use of discovery framework in Figure 1.5 demonstrates the importance of discovery services.

1.4.1 Requirements for Discovery in myGrid

1.4.2 Architecture and Stakeholders

The desired discovery architecture within myGrid also conforms to the loosely coupled architecture and its operations which we have described in Section 1.1. Characteristics of stakeholders within myGrid architecture and their implications for resource discovery can be summarized as follows:

- **Providers** of resources within the environment that myGrid operates are

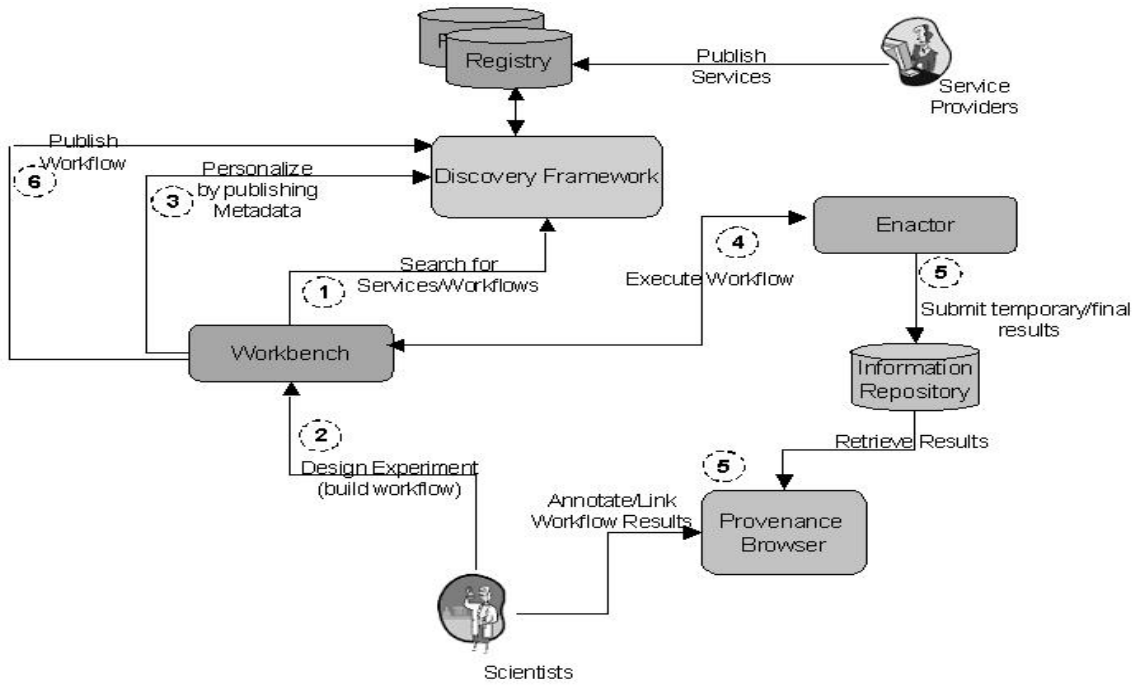


Figure 1.5: An Overview of The Architecture and Interaction of myGrid Components.

non-profit bioinformatics institutions or research labs that share their data and applications by exposing them as Services. These are shared in an **open** environment by which we mean that the services can be used by parties unknown to their provider without prior agreements on their usage. Lack of prior agreements also means that providers may change or withdraw their services anytime without notice, therefore it is solely the requestor's responsibility to adapt to such situations. The descriptions provided by these parties are generally composed of textual descriptions of service functionalities and low-level invocation interface descriptions.

- **Requestors** of resources initiating discovery in myGrid are currently limited to human-users. Users in myGrid are *"knowledgable, opinionated scientists who may be required to justify their methodologies under peer review"*[57]. Therefore they want to be involved in all the stages of service selection process. However discovery initiated by software-agents, which we term as dynamic discovery, has been identified to be needed for three particular cases in myGrid. These are:

1. Discovering and composing adaptor "Shim" services to be able to overcome syntactic incompatibilities among services brought together in a workflow.
2. Selecting among user-defined true replicas of services and swapping them for originals at times of service failures during workflow enactment to provide for improved reliability.
3. Pro-actively discovering, and suggesting the candidate services that could come before or after a certain operational step in a workflow.

Workflow building scientists (i.e. bioinformaticians or biologists) in myGrid can also act as **Annotation Providers** and generate domain specific descriptions for services in addition to the low-level invocation interface descriptions published by providers. It should be noted that users in myGrid, including those who provide annotations have minimal familiarity with recently used complex formalisms for representing domain-knowledge. Therefore they seek user-friendly tools to aid the annotation process.

1.4.3 Services in myGrid

The entities that are of interest to **users** for discovery are centred around the notion of information consuming and producing **workflow** building blocks which we term as **Operations**. Operations may correspond to:

- 'Plain' stateless Web services, by which we mean single web service operations that are described within WSDL documents and that may correspond to an operational task in the workflow.
- Stateful Web services. This category includes Soaplab [3] [79] analysis services that wrap command line tools or distributed objects and expose their stateful interaction patterns.
- Web services that have extra implicit semantics. This category includes BioMOBY services [59], which are atomic web services (i.e. services with single operations) that are registered with a specialized bioinformatics service registry named MOBY-Central. The registry imposes additional conventions to the invocation of services by acting as intermediary during interactions and mandating a specific messaging layer.
- Other Workflows. Taverna workbench enables incorporation of previously designed workflows into others. The workflows designed within Taverna are

[3] `http://industry.ebi.ac.uk/soaplab/`

represented in its modelling language named Scufl.

- Web based Web services –also known as Representational State Transfer (REST) [33] based services – which require all the information required for service invocation to be encoded in a single HTTP GET or POST request. The web based services that provide access to the SeqHound [63] biological sequence database [4] are examples of this type.
- Taverna compatible local Java objects. Taverna enables incorporation of local Java objects' methods in to the workflow design as operational steps.

While all these operations differ in their invocation mechanisms these differences are hidden from the users while they are interacting with the Taverna workbench and its associated enactor Freefluo[5]. Taverna provides a uniform representation of all above types of services under the name of **Processor** to its users. Moreover, Freefluo does not pose any restrictions on the providers of services. Instead it has been designed in an extensible way so that **Processors** for any additional type of service are developed as and when needed.

Taverna and Freefluo's unified way of access to the world of different service entities only partially alleviates the complexity of developing workflows given the large numbers of services, and the communities expectations to spentd minimum time for workflow development.

- The number of services within the bioinformatics domain is large enough to necessitate specialized components for discovery. Currently there are over 600 services that can be incorporated in to myGrid workflows [6]. The number of services keeps growing as more bioinformatics service providers supply service based access to their 'legacy' applications by use of wrapper frameworks such as Soaplab.
- In bioinformatics most workflows are developed in-house, experimented with and then they are either thrown away or modified to be turned into new experiments at a fast pace. Therefore any facilitiy that speeds up the process of experiment design is greatly appreciated.

[4] `http://seqhound.blueprint.org/`
[5] `http://freefluo.sourceforge.net/`
[6] `http://www.mygrid.org.uk/`

1.4.4 Descriptors of Services

Similar to the common discovery architecture of Section 1.1, service descriptors in myGrid contain names, textual descriptions and more structured descriptions based on an information model of services. Given that the stakeholders initiating discovery are human users, and they are provided with an common abstraction over different types of services the information model is expected to address different aspects of thiese abstracted Operations, which are:

1. Characterization of capabilities,
2. Non-functional properties,
3. Third-party assertions.

Regarding capabilities [94], services in myGrid are characterized by their *inputs* and *outputs* and a group of domain specific, user-oriented attributes,which are are:

- The overall functionality or *task* performed by a service. Like all others this attribute has no link to the invocation mechanism of a service but is quite useful to enable their selection in a biological *in silico* experiment context.
- The underlying method used by the service. Many services delivering bioinformatics tasks use certain algorithms (methods) to achieve these. Experiment designing biologists can have their own trust or choice regarding a particular method, therefore it is a useful descriptive attribute.
- The resources accessed during execution. Different services operate over different data sets, and the particular data set used by a service during execution is an important criterion for selection.
- The toolset exposed by services. As described previously in this Chapter, bioinformatics services expose bioninformatics tools or applications. Biologists do have different levels of trust or choice regarding these tools, therefore would like to select services accordingly.

Given that myGrid operates in an open environment and biologists can compose and use services in ways unanticipated by their providers, domain-specific conceptual descriptions become necessary. Similar to the example we gave in Figure 1.2, the Trader in myGrid is expected to augment the above characterization with domain-specific conceptual descriptions, where it is defined what the input,output, task, method etc. attributes of a particular service actually are.

Regarding non-functional properties, there does not exist an exhaustive list of properties expected to be addresses by a discovery system. However users in

myGrid would like non-functional service information, such as service provider details, licensing requirements, reliability, temporal and spatial availability, to be incorporated in to a service description.

Finally, the discovery system in myGrid is expected to allow publishing of 3rd party metadata assertions to a service. As described in the *in silico* experiment lifecycle, these third-party assertions would provide information on community experience or comments on using/accessing a particular service. Since each party can make assertions using their own models and vocabulary, no explicit schema should be imposed for 3rd part assertions.

1.4.5 Service Discovery Mechanism

Based on descriptors of services, the users want to perform discovery based on service capabilities, non-functional aspects, third-part descriptions. Furthermore discovery is expected to exploit the domain knowledge that has been used within the descriptions of services.

Since discovery is mostly intended to assist human-users, their information seeking behavior drives the desired forms of discovery. Users in myGrid would want to :

- Make keyword based searches on names/textual descriptions of services.
- Browse through the classification hierarchy of services based on their domain specific characteristics.
- Express search requests that may partially or fully describe a desired service with respect to the information model supported by the Trader.

The discovery framework is expected to exploit all three aspects of the Information Model:

1. Exploiting Capability descriptions:
    - Searches based on Input/Output types, "Which services accept a *protein sequence* or something general?" "Which services produce a *BLAST Report* or something more specific?" or suggestions such as "These services accepting a *protein_sequence* can come next after your workflow step producing a protein_sequence".
    - Searches based on Task, Resource, Algorithm, and Application attributes, such as "Which services perform the bioinformatics task of *sequence_alignment* or a more specific task? Which services use the

Needleman_and_Wunsch_global_sequence_alignment_algorithm or a descendant of it?".

2. Exploiting Non-Functional Aspects. We do not provide an exhaustive list of possible search requests here but examples include:
   - "Which services have the average response time of 3000ms seconds?
   - "Which services are provided by the European Bioinformatics Institute?"
   - "Which services are provided that I am licensed to use?"
   - "Which services are in my locality?"
   - "Find all alignment services except the ones provided from Japan".
3. Exploiting third party metadata aspects. Since there is no schema imposed on 3rd party metadata the questions that can be asked can not be enumerated, but examples include:
   - Which services have the highest quality rating issued by European Bioinformatics Institute?
   - Which services are marked as reliable by people from my research lab?

1.4.6 Trader Deployment

Due to the globally distributed and fragmented nature of services it is required that:

1. Descriptions should be publishable in different environments such as service registries, local information repositories or web servers.
2. Both service providers and third-party users should be able to publish and manage their own descriptions of services.
3. Since the main aim of discovery is to support users during workflow design, the Trader is expected provide a uniform interface to its discovery facilities from within the workflow workbench.

1.5 Research Questions

To address the requirements outlined in this Chapter, two discovery systems have been built during the course of ^{my}Grid project that have sought answers to the following questions:

- What are the architectural components needed for discovery?
- What is the Information Model that should be supported by the Trader?

- What is the desired role of semantic descriptions and reasoning within the discovery framework?

Our research hypothesis is that the desired discovery system in myGrid is one that combines the strengths, and avoids the weaknesses of previous approaches to discovery in myGrid.

1.6 Road Map for Thesis

The rest of the thesis is organized as follows:

Chapter 2 provides a survey of stereotypical discovery systems in different distributed environments. The necessary background information regarding our work is also given in Chapter 2 in line with the survey.

Chapter 3 describes previous efforts for service discovery in myGrid with a comparative discussion that leads to the motivation for our work.

Chapter 4 describes our discovery system, Feta.

Chapter 5 concludes the thesis with a brief overlook of which of discovery requirements have been addressed by Feta and points out the possible future research regarding discovery in myGrid.

Chapter 2

Literature Survey and Background

This chapter serves two purposes; firstly it provides a survey of discovery mechanisms in different distributed computing environments; secondly it provides background information to form a basis for the following chapters of the thesis. The chapter contains a substantial amount of information on various topics. Readers, who are particularly interested in obtaining background information for the follow-on chapters should refer to sections 2.5 and 2.4 on Semantic Web Technologies and Web Services. Those who are interested in the more specific context that our work falls into, should refer to sections 2.6 and 2.7 on Semantic Web Services and the Semantic Grid.

2.1 Scope of Analysis

The emergence of the concept of discovery dates back to the emergence of networks and distributed computing. Therefore, there are numerous examples of discovery systems in distributed environments. In this chapter we limit our analysis to **stereotypical** discovery systems in the context displayed in Figure 2.1. Our analysis will cover :

- **Discovery on The Grid** (First Generation), where mostly compute resources of diverse types are shared within closed communities and the descriptions of resource capabilities are mostly pre-defined and commonly agreed.
- **Discovery of Grid Services** (Second Generation Grid),where there are

on-going efforts to increase interoperability of closed Grid systems, by developing common information models for resource description and using web based application integration protocols for resource sharing.

- **Distributed Object Environments**, where a single type of components, software objects, are subject to discovery in closed environments.
- **Web Services**, which are largely influenced by distributed object systems regarding discovery and make use of the Web based protocols for discovering and integrating software applications in open environments.
- **Semantic Web**, which is seen as an enabling technology for seamless discovery and integration of resources on the Web. Semantic Web technologies bring in machine-interpretability to descriptions of web resources, therefore enabling increased automation of many distributed computing activities including discovery.
- **Semantic Web Services** where semantic web technologies are used to describe web services so that they can later be discovered, composed and executed by intelligent software agents. As depicted in Figure 2.1 the scope of discovery in myGrid overlaps with the Semantic Web Services area. While we describe semantic web service discovery in this chapter we will analyze discovery in myGrid in Chapter 3.
- **Semantic Grid** While classified in the Semantic Web Services area myGrid together with few other projects [24] can also be seen as a pioneer for the Semantic Grid [39]. Like the Semantic Web Services, The Semantic Grid follows a synergetic approach and aims to exploit Semantic Web technologies to build a Grid infrastructure for seamless and automated sharing and integration of diverse types of resources for the collaborative solution of science and engineering problems.

2.2 Resource Discovery on The Grid

Characteristics According to a recent definition The Grid is concerned with *"coordinated resource sharing among dynamic collections of individuals, institutions and resources named Virtual Organizations"* [36]. Examples of resources that can be shared on the Grid are CPU cycles, data storage systems, network resources, scientific instruments and databases. The distinctive feature of the Grid is its focus on large-scale, high-performance and dynamic membership of

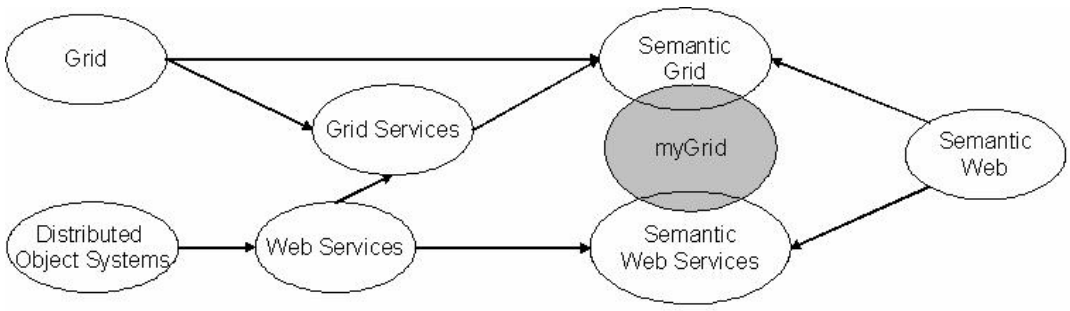


Figure 2.1: The ellipses represent the research areas that will be surveyed in this chapter. The arrows represent the evolutionary paths of research areas.

resources to a virtual organization temporarily formed to solve a problem. These features imply certain characteristics for resource discovery within the Grid.

Timely information of the system is critical to the operation of the Grid. Grid Information Services (GIS) are defined [27] to be the components that provide the essential "system information" to enable not only discovery but also resource monitoring, job scheduling activities on the Grid. GISs (i.e. Traders of the Grid environment) have three major characteristics, these are:

- Soft-State Registration: Resources on the grid may be large in size; thousands of resources can be shared in one particular Grid system at one time. In addition to their large number, resources are subject to quick change of status or failure. Hence any information regarding resources is considered old. To increase reliability of information, resource descriptions are time stamped and advertised for a limited period of time, which is known as soft-state registration.
- De-centralized organization: To cope with the large amount of resources and even larger number (i.e. tens of thousands) of requestors searching for resources, decentralized organization of discovery components that work cooperatively to answer discovery requests is inevitable in the Grid.
- Single layer capability description: Resources shared on the Grid are mostly characterized by what they are, since it suffices to be the only indicator of their capabilities (for example CPU cycles, printers, clusters). Such a single layer description is due to the fact that resources shared can expose a single capability. Whereas in other distributed systems, where a single type of resource (e.g. a web service) can expose multiple capabilities, there needs to be a layered description for the capability by means of what the resource

does.

Stakeholders Discoverers of resources on the Grid are both human users (e.g. system administrators) and software agents. Being able to deliver up-to-date descriptions about resources to the GIS in a timely manner requires increased automation in description production and submission. Such an automation can be possible in case the description producers are software agents. In most of the Grid Systems these provider agents (e.g. software agents responsible for managing a pool of machines) periodically generate and submit resource information to the GIS. Similarly requestors of resources can also be software agents such as job queue managers that is responsible for finding suitable resources for scheduling a group of jobs.

Information Models As a consequence of characteristics of the Grid GISs operate over simple and mostly specific information models that largely pre-define what type of resources can be shared in the environment and what their properties can be [10].

2.2.1 Discovery Systems in the First Generation Grid

2.2.1.1 Globus Monitoring and Discovery System 2 (MDS2)

Globus [34] is an open source toolkit that includes software services, and libraries for resource monitoring, discovery, security and file management on the Grid. The component within Globus responsible for discovery is the Globus Monitoring and Discovery Service (MDS 2). Being a typical GIS, MDS 2 proposes distributed organization for discovery components (i.e. Traders) and provides protocols for federated resource information publishing and querying. Soft-state registration is mandated for maintaining resource information within MDS components.

Information Model MDS2 [27] is based on Lightweight Directory Access Protocol (LDAP) [80] as its underlying data model. LDAP is a commonly used network directory service specification which still has a strong usage base within networked environments. LDAP provides a generic information model and a protocol for querying and manipulating it.

An example of an MDS2 resource description based on LDAP information model given in Figure 2.2. The LDAP information model is centred around hierarchically named entries which have typed attributes (see names "*hn=hostX*" and "*perf=load5, hn=hostX*"). Entries themselves are also typed by one or more

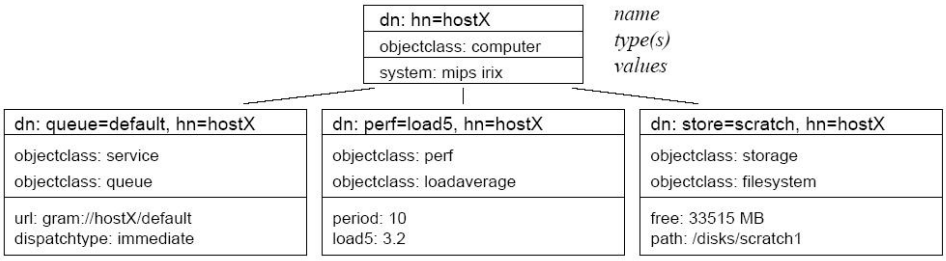


Figure 2.2: An example mapping of MDS2 auxiliary resource types to LDAP information model. Figure is taken from [27]

objectClasses which define the group of attributes an entry would have. As seen in the figure entry named "*store=scratch, hn=hostX*" has two *objectClasses* namely, *storage* and *filesystem* which indicates that the entry is supposed to have attributes *free* and *path* respectively.

Being a generic data model, LDAP does not prescribe what objectClasses should exist. In our example the objectClasses such as *computer*, *storage*, *filesystem* are imposed by MDS2. MDS2 prescribes a set of auxiliary objectClasses, influenced by an information Model of Grid Resources [10], that define properties significant for discovery.

Discovery Mechanism Discovery of resources by MDS2 Trader can be done either by browsing the LDAP entry hierarchy or by sending search requests conforming to the LDAP protocol.

Discovery via browsing is suitable for human users (e.g. system administrators) of the MDS Trader. The screen shot of an MDS2-LDAP browser is given in Figure 2.3. The screenshot displays the hierarchy of entries corresponding to Host Computers and particular resources on these computers such as processors, memory etc.

According to the LDAP protocol, during search for an entry, firstly the portion of the tree to be searched is specified along with a group of filters for desired attribute values of entities. For each matching entry the requested set of attributes are returned as a search response.

The LDAP data model and inherently the MDS2 Trader has certain limitations for discovery. Firstly, no relations between entries can be defined other than the hierarchical relationship among them. Secondly, no join operations are possible during searches. Finally, the hierarchical scheme of LDAP is optimized for

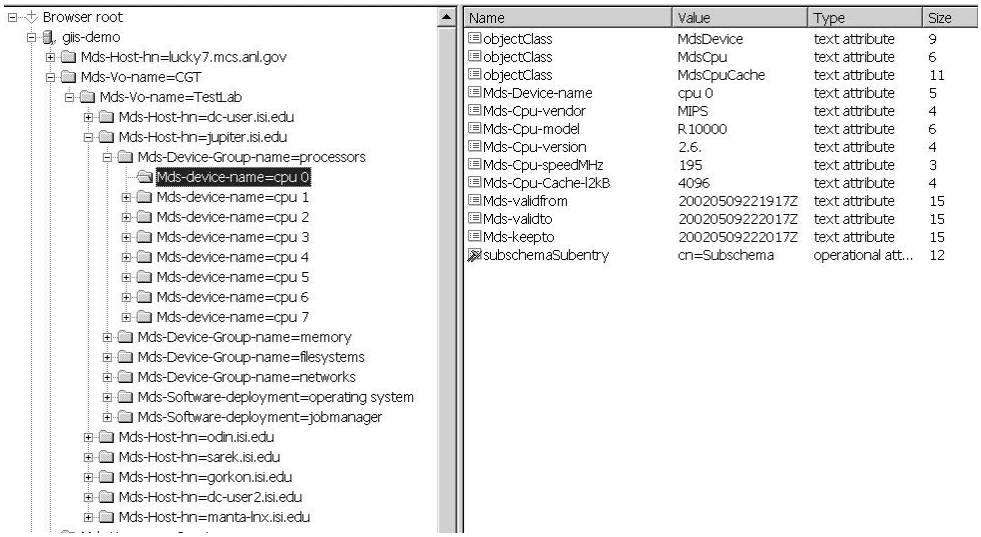


Figure 2.3: Screenshot of MDS2 Directory Browser

read and browse operations but is not equally efficient for directory tree updates or inserts.

2.2.1.2 Condor Matchmaker

Condor [22] is a well-known high-throughput Grid system developed at the University of Wisconsin. High-throughput computing aims at maximizing the amount of work done over a certain period of time rather than achieving highest-performance per time frame. Since its aim is high-throughput, being able to discover as many resources as possible for achievement of a **task** –termed **job** in Condor– is vital. The Condor system also emphasizes that resources on the Grid are distributively owned and their usage may be subject to policies set by their owners.

Data Model To be able to cope with distributive ownership, and policy requirements for resource discovery Condor provides a language called the Classified Advertisement (ClassAd) Language for describing characteristics of resources, constraints on characteristics and semantics for evaluating the constraints [75]. The ClassAd language is used for describing both resources and discovery requests for resources.

A ClassAd for a workstation with idle compute cycles is given in Figure 2.4. As seen in figure the ClassAd is a list of properties which are mappings from attribute names to expressions. In the simplest cases, the expressions are simple

```
[
        Type = "Machine";
        Name = "leonardo.cs.wisc.edu";
        DiskFree = 2048;
        Memory = 128;
        Owner = "thain";
        SpeaksFTP = true;
        LoadAverage = 0.75;
        Untrusted = {"tannenba", "wright"};
        Constraint = !member(Other.Owner, Untrusted);
        Rank = Other.RequiredMemory;
]
```

Figure 2.4: This example is a simplified version of the ClassAd that can be found in [75]

constants (integer, floating point, or string). Attribute expressions can also be more complex. These complex expressions can be built by use of arithmetic and logical operators. An example use of the logical negation operator describing the usage policy of the advertised machine is given by the *Constraint* property in Figure 2.4. This property value states that the machine is only willing to accept jobs from trusted agents.

Unlike the MDS2 system, Condor better supports distributive ownership by not prescribing what the properties can be; in other words ClassAds are schemaless except the type of attribute values (i.e. string, int).

Discovery Mechanism Both advertisements and requests are collected by a centralized matchmaker, called the Condor Matchmaker, for processing. The matchmaker evaluates expressions of both adverts and requests to find matching pairs. In Condor the stakeholders of discovery (providers of both resource classAds and job classAds) are software agents. Therefore the Condor matchmaker also supports ranking of matching results to further assist the resource selection process (see attribute Rank in Figure 2.4). Similar to soft-state registration in MDS2, resource classAds in Condor are periodically submitted to the matchmaker. Unlike MDS2, Condor does not introduce a de-centralized architecture for multiple collaborating matchmakers. However, Condor is being transformed to support a de-centralized matchmaker to allow for better scalability.

2.2.2 Discovery Systems in the Second Generation Grid

Recently, surveys [96] [53] show that there exist a vast number of Grid systems with autonomous information models and different resource sharing mechanisms. Hence concerns for interoperability have increased within the Grid community recently. The Global Grid Forum (GGF), which is a community-driven collaboration working to aid development of Grid Technologies via standardization, is the main driver behind the solutions to achieve interoperability on the Grid. The work to build the next generation of interoperable Grid systems is condensed around two tracks, these are: conception of a common information model, and a web service standards based resource sharing framework (a.k.a. Service-Oriented Grid).

2.2.2.1 Service Oriented Grid

The service-orientation effort, initiated by the Globus project and currently undertaken by GGF, is called the Open Grid Services Architecture (OGSA) [35]. OGSA proposes virtualization of grid resources as Web Services (see Section 2.4 for details on Web Services). The virtualization can also be seen as the wrapping of Grid Resources with web accessible applications. The recently conceived infrastructure that is to enable the necessary virtualization is called the Web Service Resource Framework (WSRF) [26], and the Web Service–Grid Resource couple is termed the WS-Resource. WSRF brings in additional standards to web services to be able to accommodate characteristics of Grid resources in the new Web services based framework. Examples of these standards are specifications for representing and managing the state of a Grid Resource (Service Data), specifications for representing the patterns of communication between the Grid Resource and its virtualizing Web Service and specifications for **Trader** services (Service Groups) that would allow discovery of WS-Resources.

2.2.2.2 Information Models for Interoperability on the Grid

Regarding conception of a common information model of interoperability, there are two strong candidates, the Grid Laboratory Uniform Environment (GLUE) Schema [10] and Common Information Model (CIM) [1].

GLUE schema [10], which is being developed by the DatatTAG Grid project,

is a technology-independent information model mainly targeted to support discovery and monitoring on the Grid. The GLUE Schema's technology independence allows it to be mapped to different underlying schemas. In fact the GLUE Schema has been adopted by First Generation GISs [27] [23] where it has been mapped to different underlying schemas such as the LDAP Schema [27] and the Relational Schema [23]. The GLUE Schema models Grid resources in 3 categories as: Storage Resources, Computing (Cluster) Resources, and Network resources. The schema also differentiates between resources and services which makes it compatible with the service-oriented Grid.

CIM [1], which has been proposed by an industrial collaboration called the Distributed Management Task Force DMTF, is an object-oriented technology independent schema for defining real world managed objects that occur in computer and network environments. Due to its object-orientation the managed objects within CIM are not only defined by their attributes but also by the management operations they support. Similar to the GLUE Schema, CIM also has the notion of Services within its model. Compared to GLUE Schema CIM model has a broader scope and is more elaborate. It is possible to make mapping of all elements in the GLUE Schema to the CIM elements. Recently CIM has been endorsed by the GGF to be used for the realization of the OGSA vision.

2.2.2.3 Globus Monitoring and Discovery System 3 (MDS3)

The product of efforts on increased interoperability that falls in our context is the latest version of the Globus Monitoring and Discovery System, MDS3 [2]. MDS3 is a service-based re-implementation of MDS2. The principles behind MDS2 such as soft-state registration, distributed Traders are preserved within MDS3. The difference between two systems is that resource providers and requestors interact with the MDS3 Trader, which is provided as a Grid Service, via web-service protocols instead of the LDAP protocol used in MDS2. In addition, resource descriptions conform to the GLUE Schema, which is mapped to XML Schema as the underlying model. Descriptions kept in XML format are queryable by XML query languages such as XPath.

2.2.3 Remarks on Discovery on the Grid

The stereotypical Traders on the Grid we have chosen for surveying in this thesis have the following common characteristics (also summarized in Table 2.1) (1) single layer information models for describing what are the resources that are being shared, (2) simple attribute value based underlying models to store and query resource descriptions, (3) effective matching methods used during discovery, (4) support for distributed soft-state resource publishing, and (5) Trader cooperation to cope with scalability, high-performance and timeliness requirements of the Grid.

While these traders such as the Condor matchmaker and the MDS2 system are tailored for the Grid, they may not provide desired discovery facilities in an open environment where resources exhibit multiple capabilities and no single description for them exists. In such environments richer descriptions of resources and a smarter service selection process would be needed to achieve seamless integration. While there exists increased automation in terms of description and discovery on the Grid, the discovery process is performed with respect to predefined agreements between providers and requestors.

2.3 Distributed Object Discovery

Characteristics Another environment for distributed computing that is frequently acknowledged within the discovery context is the distributed object based middleware infrastructures such as the Object Management Group(OMG)–Common Object Request Brokerage Architecture (CORBA) [68]. CORBA allows interoperability among distributed software objects by allowing their remote invocation via a broker and by providing a group of auxiliary functions that are commonly needed in a distributed environment (e.g. discovery, lifecycle management, security for objects). Among the auxiliary functions object discovery is provided via the OMG Naming [69], and Trading [70] services.

2.3.1 OMG-Naming Service

Information Model and Discovery Mechanism The OMG Naming service allows providers to associate names with CORBA objects and allows requestors to locate those objects via lookup. It employs a hierarchical naming scheme. The

```
Interface sequenceAnalysis{
    void performAnalysis (in string program_name,
                          in string database_name,
                          in string query,
                          out string result)
}
```

Figure 2.5: An Example IDL Description.

Naming service is not further described here.

2.3.2 OMG-Trading Service

Information Model. Discovering objects by their properties is provided by the OMG Trading service [70]. The OMG Trader allows providers to publish descriptors for objects called Service Offers, which conform to certain Service Types. The Service Type represents the information needed to describe the object. It is composed of:

- The Interface Type of the object (i.e. object capability), which is a pointer to the computational signature of the object expressed in the CORBA Interface Description Language (IDL). A sample IDL description of an object's interface is given in Figure 2.5. According to this IDL description the object has an interface named *sequenceAnalysis* and an operation named *performAnalysis* that has three input parameters and one output parameter all of type *string*.
- Zero or more property types for the object, which are composed of name, type and mode triples (e.g. name=loadPercentage, type=int, mode=mandatory). Property types are typically used to represent non-functional aspects of the object.

The Service Offers submitted to the Trader contain the adhered Service Type name, the reference to the the object and zero or more property values.

Properties of objects advertised in the OMG Trader can also be dynamic. Dynamic properties are evaluated at discovery time to provide the most up-to date information about an object. This aspect of OMG Trader is similar to the timely and up to date system information requirements of Grid Information Systems.

The Service Type mechanism is the only modelling element of the OMG Trader to describe the capability of an object (i.e. IDL) and its other aspects (e.g. non-functional, provider related)

Discovery Mechanism. The OMG Trader provides interfaces for registration and object search. Via the registration interface, providers of objects can register Service Types or they can register objects that adhere to a certain Service Type. Via the search interface it is possible discover objects that implement a particular IDL interface (e.g. *sequenceAnalysis*) and has certain values for the properties advertised in a Service Type (e.g. loadPercentage= 50). The OMG Trader specification also introduces the concept of linking to form federates of Traders.

Stakeholders Requestors for objects within the CORBA environment are both human-users, and software agents.

The OMG CORBA specification has a *Dynamic Invocation Interface* which provides mechanisms for making dynamic calls to objects discovered via the Trader Service. Even though dynamic discovery and invocation are possible, it is rather hard for call making programs to interpret what the operations of the newly discovered object do and what the parameters actually correspond to, due to lack of semantics associated with methods and their parameters. Hence dynamic discovery and invocation have been used in a limited fashion in CORBA, where objects were discovered based on their invocation interface via the Trader, and all objects with the same interface are assumed to have the same semantics for the operations and parameters. This way the dynamic aspects of the discovered objects are their providers, and their location.

2.3.3 Remarks on Distributed Object Discovery

Our discussions in the previous subsection underline the most important characteristics of object discovery systems which chiefly suppport object lookup via unique names and discovery via object types and properties. The IDL based typing of objects is far from being a meaningful description of the object's capability to both humans and machines. Due to this fact use of such Traders has been limited to closed environments where providers and requestors are aware of each other and given the IDL each party can interpret the capability of an object based on their prior agreements.

Having described the discovery of Distributed Objects where objects are shared

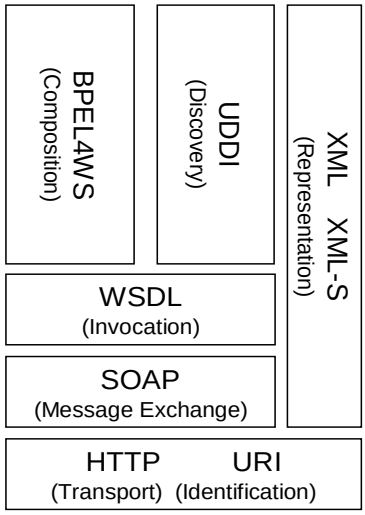


Figure 2.6: Web Services Enabling Standards

in a closed environment where mostly everything is commonly agreed upon and a single binary protocol is used for communication, we will proceed to a vision that enables sharing of software applications in an open heterogeneous environment namely the Web Services.

There exists a big debate on the similarities and differences of Web Services and Distributed Objects [92] [38]. However, there is no doubt that Distributed Object systems have given way to Web Services. Furthermore regarding discovery we observe similarities between the two, which we will point out in the summary section at the end of this chapter.

2.4 Web Services

Characteristics Web Services [93] are the recent paradigm for distributed computing which uses the World Wide Web and its associated protocols as a medium for sharing, and integrating software applications.

According to the World Wide Web Consortium's (W3C) definition, a Web Service is *"a software system identified by a Uniform Resource Identifier, whose public interfaces and bindings are defined and described using the eXtensible Markup Language (XML). Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols"* [93].

W3C's definition of Web Services contains implicit references to the enabling standards (i.e. languages and protocols) of this distributed computing paradigm presented in Figure 2.6.

All standards displayed in Figure 2.6 are based on the foundational markup language XML and its schema XML-Schema. Not surprisingly, web services' foundational protocols for transport of information are web based transport protocols such as HTTP or FTP. Based on these two foundations the standards can be summarized as follows:

- Simple Object Access Protocol (SOAP) is an XML-based protocol that defines a framework for passing messages between systems over the internet. SOAP typically is used to establish remote procedure calls between web services and their clients over the web.
- Web Services Description Language (WSDL) is an XML based language for describing web services as a set of endpoints operating on messages.
- Universal Description Discovery Integration-UDDI is a Directory Model to enable discovery of Web Services.
- Business Process Execution Language for Web Services (BPEL4WS) is a standard, rapidly proceeding to become the de facto standard for specifying compositions of web services as process models.

Within the Web Services environment the aforementioned Provider-Trader-Requestor Architecture (see Figure 1.1 of Section 1.1) and its operations are adopted as the Web Services Model [87]. Since the focus of this chapter is on discovery we find it essential to provide detailed analysis of Web Services standards for description and discovery namely WSDL and UDDI in the following sub-sections.

2.4.1 Web Services Description Language

WSDL [20] is the W3C recommended language for describing interfaces of web services. A sample WSDL description of a well known bioinformatics sequence alignment service BLAST (Basic Local Alignment Search Tool) is given in Figure 2.7. The WSDL specification describes service interfaces in two layers as abstract and concrete.

- The **abstract layer** enables description of service *operations* in terms of input and output *messages*. The example in figure 2.7 describes a single operation named *searchSimple* delivered through the location ***http:***

```
<definitions name='Blast'>

  <message name='searchSimpleIn'>
    <part name='program' type='xsd:string'/>
    <part name='database' type='xsd:string'/>
    <part name='query' type='xsd:string'/>
  </message>

  <message name='searchSimpleOut'>
    <part name='Result' type='xsd:string'/>
  </message>

  <portType name='Blast'>                                          Abstract
    <operation name='searchSimple'
               parameterOrder='program database query'>

      <documentation>
       Performs Blast on a given sequence with using designated
       database
     </documentation>

     <input name='searchIn' message='tns: searchSimpleOut '/>
     <output name='searchOut' essage='tns: searchSimpleOut '/>

    </operation>
  </portType>

  <binding name='Blast' type='tns:Blast'>
    <soap:binding
          style='rpc'
          transport='http://schemas.xmlsoap.org/soap/http'/>
    <operation name='searchSimple'>
      <soap:operation soapAction='searchSimple' style='rpc'/>
      <input name='searchIn'>
        <soap:body encodingStyle=' http://schemas.xmlsoap... '/>
      </input>
      <output name='searchOut'>
        <soap:body encodingStyle=' http://schemas.xmlsoap... '/>
      </output>                                                    Concrete
    </operation>
  </binding>

  <service name='BlastService'>
    <port name='Blast' binding='tns:Blast'>
      <soap:address location='http://xml.nig.ac.jp/xddbj/Blast'/>
    </port>
  </service>

</definitions>
```

Figure 2.7: A Sample WSDL Description for a Sequence Alignment Service

//xml.nig.ac.jp/xddbj/BlastDemo. The operation has input and output messages named *searchSimpleIn* and *searchSimpleOut* respectively. The input message is defined to be composed of three parts named, program, database and query. Each of these parts is a parameter to the service.

- The **concrete layer** maps abstract operations and messages to physical endpoints in terms of *ports* and *bindings*.

WSDL has a role similar to that of IDL in distributed object computing platforms. However the description contained in WSDL documents is not limited to the invocation signature like the case of IDL. WSDL documents also contain information on how to bind to a particular service via use of different web-based protocols for message exchange (e.g. SOAP, MIME) and transport (e.g. mailto, http, ftp, etc.) In WSDL, operations and messages can be bound to multiple physical end points implementing different protocols for communicating with the service. Although no single protocol is mandatory, SOAP is the most commonly used XML based protocol [15] that enables exchange structured information over web-based transport protocols (e.g. HTTP).

However, similar to IDL, WSDL also suffers from lack of semantics. The WSDL specification does not allow metadata to be attached to the port types, operations, and messages to better describe their meaning. Following from our example in Figure 2.7, though the names of these message parts happen to be descriptive of their nature to a human, the WSDL specification has no means to describe what these parameters actually are.

2.4.2 Web Services Discovery with UDDI

Returning to our area of interest in this survey, the web services de facto standard for discovery is UDDI. It is a service registry specification jointly proposed by a group of industrial collaborations [7]. Being oriented towards e-business, UDDI allows XML based registration of businesses and their Services (see the XML fragment in Figure 2.9). UDDI provides three types of information on services these are:

- white pages, including address, contact, and known identifiers;
- yellow pages, including industrial categorizations based on standard taxonomies;
- green pages, containing the technical information about services that are exposed by the business. Green pages include references to specifications

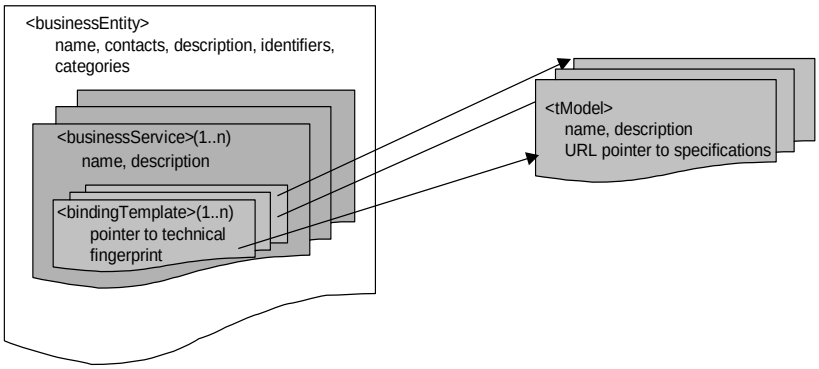


Figure 2.8: UDDI Information Model

for Web Services, as well as support for extensibility via pointers to various file and URL based external discovery mechanisms if required.

The UDDI Information Model, given conceptually in Figure 2.8, identifies four core types of information. The *businessEntity* element is a container for white pages information. The *businessService* element is used to represent a group of services provided by a *businessEntity*. The yellow pages and green pages functionality is achieved through the use of the use of *bindingTemplate* and *tModel* elements. The use of tModels –abbreviation for Technical Models– in UDDI is two-fold [25]:

1. They define the technical fingerprint of services (i.e. green pages information). UDDI suggests that tModel entities can be representatives of technical descriptions (e.g. WSDL) in the registry.

2. They provide abstract namespace references to be used in the categorization of businesses, and business services (i.e. yellow pages information). The UDDI specification enables its users to use tModels to represent taxonomies such as North American Industrial Classification Scheme (NAICS) [3] for businesses or United Nations Standard Products and Services Code (UNSPSC) [6] classification for products.

It should be noted that UDDI specification does not impose any restrictions on the usage of the tModel entities, so the correspondents of tModels are not explicitly specified. tModels are references stored in the UDDI registry which are representatives of various technical specifications which reside outside the UDDI

Registry. So UDDI is not a repository for specifications but is a notice board announcing their existence.

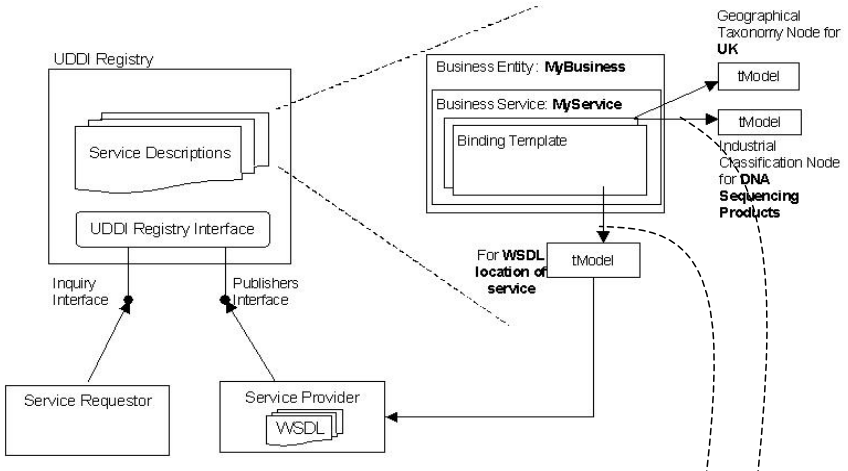
We will follow our previous example of the BLAST service and describe how such a service could be registered in the UDDI registry. Before proceeding with the example, we should note that web services operate in an open heterogeneous environment where they need rich descriptions that contain not only invocation interface description but also several non-functional properties [72] such as settlement methods, temporal/spatial availability, delivery methods, security facilities, Quality of Service by means of performance, reliability and so forth. Therefore, our example UDDI description of the BLAST service will include not only reference to its invocation interface description (WSDL), but also its provider's geographic location, the functionality provided by the service by means of an industrial classification and its quality rating. The only mechanism provided by UDDI to describe service properties is the tModel references.

Figure 2.9 displays how a service description for the desired service might be registered in the UDDI registry both from a conceptual view and its concrete XML representation. Using the industrial classifications already registered in the UDDI registry we can associate a service with the UNSPSC classification code of "DNA Sequencing" products. Using the code for UK in ISO3166 Geographical taxonomy we can associate the service with its geographical location. If a classification for quality ratings has been registered with the UDDI registry we might associate it with our service by referencing its tModel key. Additionally a tModel for the WSDL description of the service can be generated as a pointer to the WSDL document on the providers site.

Discovery Mechanisms UDDI Registry provides two APIs; one for publishing of XML Service descriptions conforming to its information model, and one for making inquiries about services by their:

- **Names**: It is possible to locate businesses and their services by key word based searches on their names published in the UDDI registry. (i.e. whitepages lookup)
- **Properties**: It is possible to locate services that are associated by a particular tModel. (i.e. yellow and green pages lookup)

.




```
<businessService businessKey="..." serviceKey="...">
  <name>BLASTervice</name>

  <description xml:lang="en">
    Basic Local ALignment Service
  </description>

  <bindingTemplates>
    <bindingTemplate bindingKey="..." serviceKey="...">

      <accesssPoint URLType="http">
        http://xml.nig.ac.jp/xddbj/BlastDemo
      </accessPoint>
      <tModelInstanceDetails>
        <tModelInstanceInfo tModelKey="[tModel Key for Service Interface]">
          <instanceDetails>
            <overviewURL>
              http://xml.nig.ac.jp/xddbj/wsdl/Blast.wsdl
            </overviewURL>
          </instanceDetails>
        </tModelInstanceInfo>
      </tModelInstanceDetails>
    </bindingTemplate>
  </bindingTemplates>

  <categoryBag>
    <keyedReference tModelKey="[tModel Key for UNSPSC]"
                    keyName="Deoxyribonucleic acid DNA sequencing products"
                    keyValue="41105600"/>
  </categoryBag>
</businessService>
```

Corresponds to

Figure 2.9: The XML Fragment Corresponding to the UDDI Entry for the Sequence Alignment Service

The tModel based search mechanism provided by UDDI is similar to the "Service Type" based searches of OMG Trader. UDDI's tModels, which are encouraged to represent WSDL descriptions, correspond to OMG Trader's Service Types, which are representatives of IDL descriptions. (See Table 2.1 for a comparison of these systems). Also similar to the OMG Trader, the most recent version of the UDDI specification (Version 3) brings support for federated UDDI nodes. Since this specification is fairly new this feature has not been implemented in discovery systems that extend UDDI or are based on it.

Stakeholders in Web Service Discovery Even though UDDI descriptions indirectly relate services with properties, these descriptions are ambiguous for software agents and even for humans because:

- UDDI allows registration of services that are not restricted to computational web services. This causes the UDDI Information Model to be too generic so that it does not have explicit links to other Web Services standards like SOAP, WSDL in its specification.

- Associating a service with a tModel merely says that the service is related to the entity represented by the tModel. But this association does not describe the nature of the relationship for our example case we can not specify whether the service *selling* DNA sequencing products or *providing training on* them or has some other relationship to them.

- The taxonomies registered in UDDI represent a hierarchy where each node is uniquely identified by a tModel key. Neither relationships among nodes of the hierarchy nor their properties can be reflected and used during service description and discovery. For example, if the service description references the tModel corresponding to "London" in the ISO hierarchy this service will not be in the result set for our search since the knowledge of London being in UK is not stored in UDDI.

- There is no direct way to attach properties to services.

Therefore UDDI based descriptions is not suited to allow dynamic discovery of services by software agents. It is rather intended for human-users who can search for services based on key word search or industrial classifications based searches.

2.4.3 Enhanced Web Service Discovery

Several research groups have drawn attention to the fact that UDDI has limitations for discovery, and have proposed extensions to it rather than using it in conjunction with external discovery facilities.

2.4.3.1 UDDI Enhancements

In [76] an extension to the UDDI Information model has been proposed. The extension element named *qualityInformation* will enable the information to explicitly hold Quality of Service information like scalability capacity, performance, throughput, reliability on services.

In [9] an extended UDDI registry, named UDDIe, has been developed by the University of Cardiff as part of the UK e-Science initiative. The extended registry allows attachment of metadata to UDDI descriptions in the form of user-defined **named properties** (e.g. Service's *quality rating* is *5**). Additionally, the idea of service leasing or temporary registration has been introduced. Service leasing is similar to soft-state registration in the Grid Information Services and aims to avoid service descriptions in the registry becoming stale.

2.4.4 Remarks on Web Service Discovery

Service Registries provide detailed specifications for registry content management, replication, publish/subscribe based notification of registry updates, multi registry interactions and custody and ownership of service advertisements. In short, they provide the specifications for the Universal Business Registry with reliable and well-maintained content. While having strong commitment to these aspects, service registries do not provide much more functionality than early name/directory services such as LDAP or CORBA. UDDI provides a technical fingerprint, tModel, based lookup mechanism which is simplified for the sake of genericity which makes UDDI usable in multiple domains. The interpretation of tModels is not addressed within the UDDI specification. It is rather left to different domain specific search facilities that could be augmented with UDDI.

To this end we have described the UDDI approach to discovery its limitations and immediate solutions to these limitations. Work on discovery of web services is certainly not limited to UDDI.

Based on the observations that service registries and WSDL lack support for

metadata and lead to ambiguous descriptions, recent efforts focus on development of unambiguous knowledge rich service descriptions and smarter service selection procedures. These recent efforts are termed the Semantic Web Services (SWS). Prior to describing SWS we find it necessary to describe the Semantic Web Technologies that are the key enabler of SWS.

2.5 Semantic Web Technologies

In parallel to the emergence of Web Services paradigm the initial design of the World Wide Web has been revised by its inventor [14] to cater for increased automation. A new vision termed "The Semantic Web" has been conceived. The aim of the semantic web research activity is to transform current web content, which is mainly targeted to humans, into a machine understandable format. Once web content becomes machine understandable the number of activities performed by unattended software agents on behalf of humans will dramatically increase.

Figure 2.10 displays the big picture of the semantic web, also known as the Semantic Web Food Chain [29]. Within this vision of increased automation the prime consumers of web content are software agents, rather than human users. In order for the agents to understand web content, data in it needs to be formally represented. Creation of formal data is achieved via the development of ontologies, and annotation of web pages using those ontologies. Ontologies are formal specifications of vocabularies used to describe a specific domain. Ideally there would be a single ontology for each domain of information on the web, however such an expectation is unrealistic, and it is anticipated that there will be multiple ontologies which can be articulated and linked to each other by the help of Articulation Toolkits (see Figure 2.10).

The added value of introducing agents to the web comes from their ability to integrate large amounts of highly distributed information (that comes from the annotations) on the web into metadata repositories (see Figure 2.10), and to be able to infer new information that has not been explicitly stated within the annotated web resources. This way agents act as knowledge intermediaries for humans (1) by integrating highly distributed information on the web, and (2) by mimicking their reasoning capabilities by the help of inference engines.

Human users are at the top of the Semantic Web food chain that consume the composite knowledge generated for them. To realize such a food chain, a

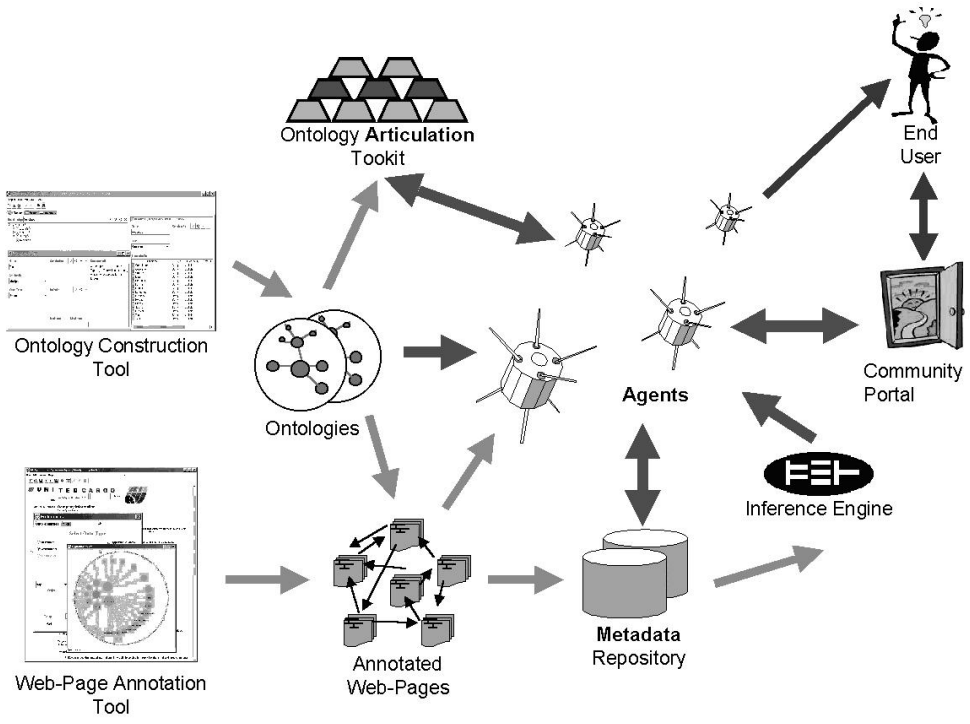


Figure 2.10: The Information Food Chain for Applications on the Semantic Web. This figure is taken from [29].

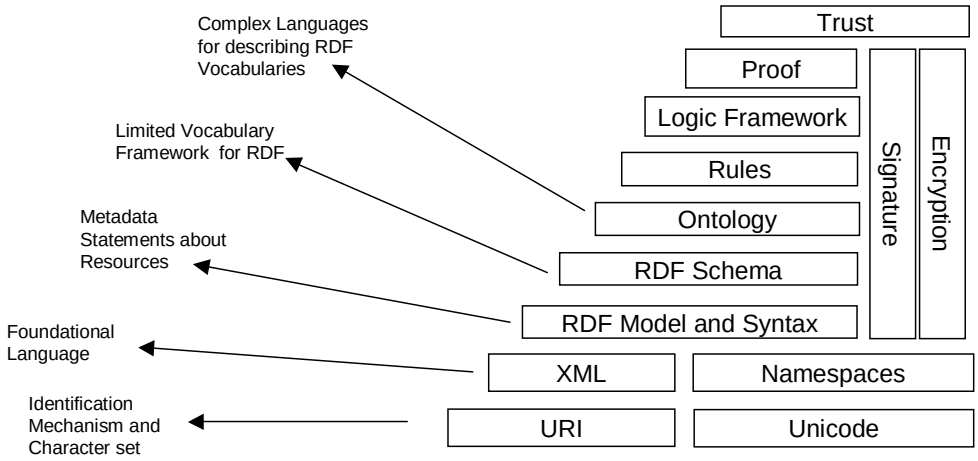


Figure 2.11: The Semantic Web Languages

stack of data, metadata, information and knowledge representation formalisms is necessary. The group of formalisms that assist each step in the semantic web food chain is called the semantic web language stack shown in Figure 2.11. At the lowest layer of the stack resides the URI and Unicode standards that enable identification and encoding of resources such as web pages all over the web written in different natural languages. Above them is XML which is a foundational markup language which also provides unique naming for markup tags via its namespace mechanism. XML also acts as the serialization language for languages at the higher levels of the stack. RDF is the next language in the stack that provides the mechanism to state facts (i.e. provide metadata) about the resources. RDF's schema language RDF(S) is the provider of the mechanism to build vocabularies that are used in RDF statements. RDF(S), which is itself a primitive ontology language, provides the foundations for more sophisticated ontology languages that allow machine interpretable specification of knowledge such as the recent W3C Recommended web ontology language OWL. The Rules, Logic, Proof and Trust layers above ontologies are currently in their development stage.

Within our survey we will be focusing on three levels of the Semantic Web Language Stack. These are RDF, RDF(S) and Ontology layers.

2.5.1 Resource Description Framework RDF

The Resource Description Framework (RDF) [58] is a World Wide Web Consortium (W3C) recommended metadata standard primarily developed to describe Web resources. Within the RDF Data model metadata about a resource is presented by RDF Triples. Triples are in the form of statements composed of a Subject, a Predicate, and an Object;

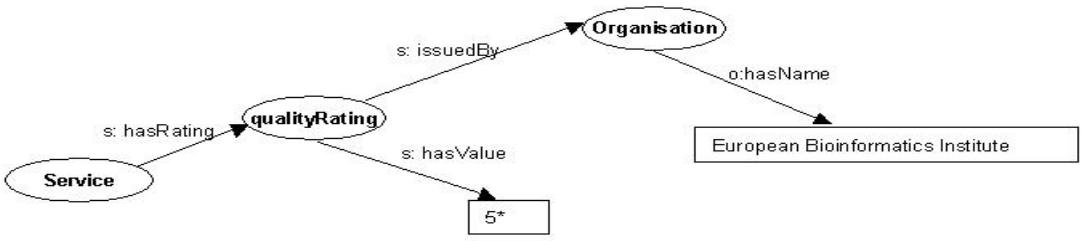


Figure 2.12: A Sample RDF Graph Representing a Group of Statements About a Web Service. The unique identifiers of resources (shown with ellipses) are replaced with names for the sake of readability.

To be more precise, RDF is based on the idea of identifying resources using web identifiers (called Uniform Resource Identifiers, or URIs) and providing metadata about resources (i.e. subjects) in terms of simple properties (i.e. predicates) and property values (i.e. objects).

Using simple statements (i.e. triples) it is possible to build a graph of nodes and arcs, representing the resources and their properties and values (See Figure 2.12). As seen in the figure we can state facts about a web service with the following group of statements "The service (identified by its WSDL URL) has a quality rating which has been issued by an institution (identified by its web page URL) named European Bioinformatics Institute and has value 5*" in RDF.

Within RDF here are no restrictions on the statements, other than the fact that resources and properties need to have URIs. This is similar to the schemaless description language of Condor Matchmaker of the Grid with the difference that the describing statements of a resource are uniquely identified since RDF aims to operate in an open environment like the Web.

RDF Data can be queried via query languages that allow expressing graph patterns as queries. A detailed description and discussion of these languages is beyond the scope of this thesis [74]. However it can be noted that among the

existing languages for querying RDF data, RDQL stands out as the most widely implemented and used language. RDQL is a SQL-like syntaxed query language that allows specification of a graph pattern to be matched against the graph of RDF data. Following our example it is possible to pose the question "Find me the names of all institutions who have issued a quality rating to a particular Service, and have issued the value 5* to the rating " as an RDQL query.

2.5.2 RDF Schema

RDF metadata statements about web resources are of limited value if they are not stated using a controlled vocabulary. RDF user communities also need the ability to define the vocabularies (terms) they intend to use in those statements, specifically, to indicate that they are describing specific kinds or classes of resources, and will use specific properties in describing those resources.

Following our example in Section 2.4.3, the service can be described using classes such as *ws:Service* and using properties such as *ws:hasRating*, *ws:issuedBy*. Similarly, people interested in describing the organizations would want to use classes such as *o:Organization* or *o:LicensedOrganization* and use properties such as *o:hasName*, *o:hasAddress* to describe them. RDF itself provides no means for defining such application-specific classes and properties.

Instead, such classes and properties are described as an RDF vocabulary, using extensions to RDF provided by RDF Schema (RDFS) [16]. RDFS does not provide a vocabulary of application-specific classes and properties. Instead, it provides the facilities needed to describe such classes and properties, and to indicate which classes and properties are expected to be used together (for example, to say that the property *o:hasAddress* will be used in describing an *o:organisation*). In short, RDFS provides a type system for RDF. Additionally, it allows classes and properties to be to be organized in a hierarchical fashion: for example a class *o:StandardsOrganization* might be defined as a subclass of *o:Organization*, or the property *o:hasAddress* can be defined to be a sub-property of *o:hasContactDetail*.

The sub-class and sub-property relationships are transitive, and the transitive closure of such classes can be exploited during querying of RDF statements. This is also known as RDF(S) reasoning. As an example suppose that the description of our example service is made in RDF which describes it to be "a service located in the city of London". And lets further assume that this RDF statement conforms to a schema which defines the class *London* as a sub-class of *UK*, meaning

"London being a city in UK". In such a case our queries for "services located in UK" would also return the service located in London.

Even with the use of RDF(S) there is still knowledge about web resources that we cannot express in the descriptions. RDF(S) is limited to describing sub-class/super-class relationships but there maybe other relations among terms that we would like to express; such as "Organisations located in the UK who have supplied at least one web service rating are UK Rating Organizations". This is a kind of knowledge that needs to be expressed with languages that are more expressive than RDF(S). These languages are ontology languages that reside on the layer above RDF(S) in the Semantic Web Language Stack (see Figure 2.11).

2.5.3 Ontologies

To be able to provide more expressive descriptions of web resources the knowledge regarding those resources need to be specified in unambiguous machine-interpretable forms. At this point ontologies take stage. Several definitions of an ontology exist one of them is as follows;

> An ontology may take a variety of forms, but necessarily it will include a vocabulary of terms, and some specification of their meaning. This includes definitions and an indication of how concepts are inter-related which collectively impose a structure on the domain and constrain the possible interpretations of terms [89].

Another well-known definition by Gruber is:

> Ontology is a specification of conceptualizations, used to help programs and humans share knowledge [41].

Ontologies represent knowledge in a domain via concepts, relations, instances and axioms [81].

- A concept represents a set or class of entities or 'things' within a domain. For example a DNA_Sequence is a concept within the domain of molecular biology. Concepts may either be (1) Primitive concepts which only define the necessary conditions (in terms of their properties) for being a member of the class that the concept represents (e.g. The concept *DNA_Sequence* can be defined to *have sequence units of Nucleic_Acid*. This concept definition expresses that all DNA_Sequences have sequence unit of Nucleic

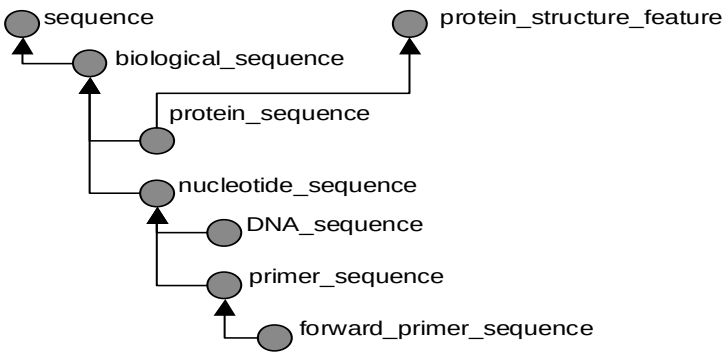


Figure 2.13: A Sample Sub-Class Hierarchy of Concepts in the myGrid Domain Ontology

Acid, however not everything having a sequence unit of Nucleic Acid is a DNA_Sequence) or (2) Defined concepts which define properties that are both necessary and sufficient for things to be a member of the class. An example of a concept described using other concepts can be the description of *Enzyme* as a composite concept using concepts *Protein* and *Reaction*, joined with the relation *catalyses* - to define Enzyme as a Protein which catalyzes a Reaction.

- Relations describe the interactions between concepts or a concept's properties. Using relations concepts may also be organized into taxonomies (e.g. Sub-Class, IS-A relationships). An example of a taxonomy of classes with respect to sub-class relationships in the myGrid domain ontology is given in Figure 2.13.

- Instances are the 'things' represented by a concept. An example of an instance can be one of the inputs of our example BLAST Service, which can be defined to be an instance of the concept DNA_sequence_data. Normally instances are not specified within an ontology. Instances of concepts are stored in Knowledge Bases.

- Axioms are used to constrain values for classes or instances. An example of an axiom from the myGrid ontology can be one that defines classes *aligning* and *retrieving* to be disjoint.

2.5.4 Ontology Languages

Ontologies are specified in Knowledge Representation Languages. These languages can be object (frame)-based, logic –particularly Description Logic (DL)– based, or a combination of both.

- Frame-based representations [51] have the notion of frames or classes which represent concepts of the ontology. Each frame has an associated collection of slots or attributes which can be filled by values or other frames. Attributes of frames are local so they are only applicable to the frames they have been associated with. Frames are popular for their human-friendly approach to modelling and its similarity to other object-based modelling approaches such as Unified Modelling Language (UML) [12]. Frame based representations have later been extended to be able to represent fragments of First Order Logic. These extended formalisms, such as F-Logic [50] or Operational Conceptual Modelling Language OCML [66], are also known as classical ontology languages.
- Description Logics (DL) are logics with well-defined semantics. DLs describe knowledge in terms of concepts and relations (roles). The major characteristics of Description Logics are that:
  (1) They allow the describing of new concepts, known as defined concepts, from atomic ones using concept constructors, which can be relations or operators (e.g. conjunction, disjunction). This way the knowledge model is built up from small pieces in a descriptive compositional way.
  (2) They are equipped with decidable reasoning procedures that can be used for consistency checking of defined concepts, and for subsumption checking between defined concepts.

DL based Semantic Web ontology languages and their relationships are given in Figure 2.14. Following from the figure:

- DAML-ONT [60], developed as part of the US DARPA Agent Markup Language, is a simple language for expressing more sophisticated RDF class definitions than permitted by RDF(S).
- OIL [32] (the Ontology Inference Layer), developed by a group of European researchers, is an early ontology language for the web that integrates frame based primitives with reasoning capabilities of Description Logics.
- DAML+OIL [44] is an expressive description logic formed as a result of a merge of DAML-ONT and OIL. DAML+OIL extends the vocabulary of

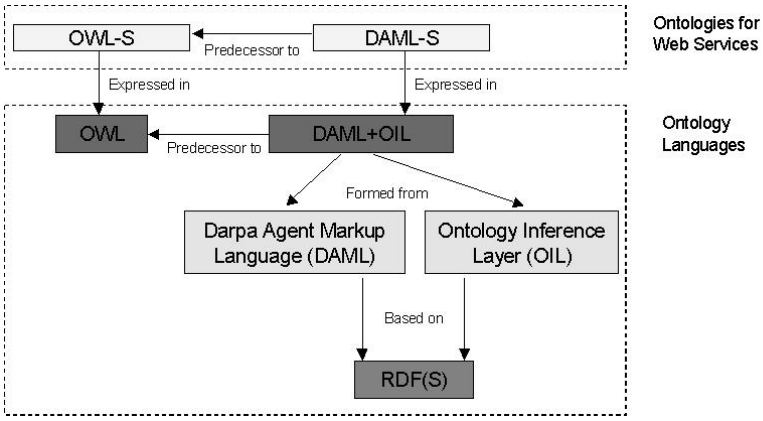


Figure 2.14: A closer look at the Ontology Layer of the Semantic Web language Stack

RDF(S) for describing properties and classes: DAML+OIL classes can be names (i.e. atomic) or expressions (composite). Additionally a variety of constructors are provided for building class expressions (e.g. intersectionOf, unionOf and complementOf). DAML+OIL also support a set of axioms (e.g.subClassOf, sameClassAs, disjointWith) that can be used to define subsumption or equivalence or non-equivalence among classes, which may be primitive or defined.

- The recent W3C recommended language Web Ontology Language (OWL) [61] is derived from its pre-decessor DAML+OIL by incorporating the lessons from its design and use. OWL is different from DAML+OIL in two aspects; firstly OWL does not allow qualified number restrictions to be made on values of properties that a class has, and secondly it allows properties to be symmetric. Both DAML+OIL and OWL are tightly integrated with RDF(S). RDF(S) is used to express their machine readable specification. More detailed descriptions of expressivity of these web ontology languages can be found in [44] [46].

2.5.5 Reasoning

As we have previously mentioned, Description Logics are equipped with reasoning capabilities which are mainly used during :

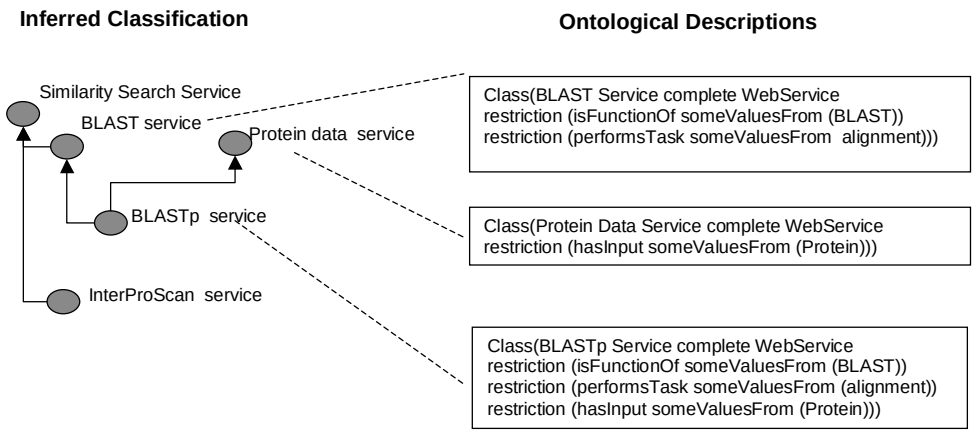


Figure 2.15: The inferred classification hierarchy based on separate class descriptions.

- Ontology development to (1) check consistency between the defined concepts and axioms within the ontology, (2) deduce implied subsumption relations between concepts in the ontology. Subsumption checking allows automated generation of an inferred classification hierarchy among defined concepts in the ontology. The left hand side of Figure 2.15 displays such a classification from myGrid that is generated by the DL reasoner using the ontological descriptions of different types of bioinformatics services, displayed on right hand side of the figure, defined in the myGrid domain ontology. Ontology development is generally done by use of ontology editors. Figure 2.16 displays the screenshot of the ontology editor Protégé [52] displaying the myGrid bioinformatics domain ontology. In the screenshot both the subsumption hierarchy of concepts and restrictions on concepts can be seen. Ontology editors [52] [13] integrated with DL reasoners are particularly useful for developing and maintaining large and complex ontologies with multiple authors.
- Ontology deployment. Besides providing the common vocabulary and metadata framework, the higher level expressive power of ontologies are unleashed when reasoners are used for answering questions within Semantic Web enabled applications. A questions such as "Which service accepts an input of ontological type *DNA_sequence*?" can be answered by a reasoner if the question is formulated as a defined class which is a service that has

the hasInput property with the restriction of property value being an instance of DNA_Sequence. Such a description could then be processed by the reasoner and placed in a service classification hierarchy like the one in Figure 2.15. The sub-classes that are classified under our defined concept (i.e. our question) would be the result of the question.

DL Reasoners employ optimized tableaux algorithms for subsumption, and consistency checking among concepts. Examples of widely-used DL reasoners are FaCT [43] and RACER [42]. The performance of such tools can be limited based on the size of the ontology and the associated knowledge base they operate on. As the expressivity of the ontology language increases, the cost of performing reasoning over it increases as well. The least expressive Description Logic *ALC* has reasoning procedures with exponential time, and space complexity [43].

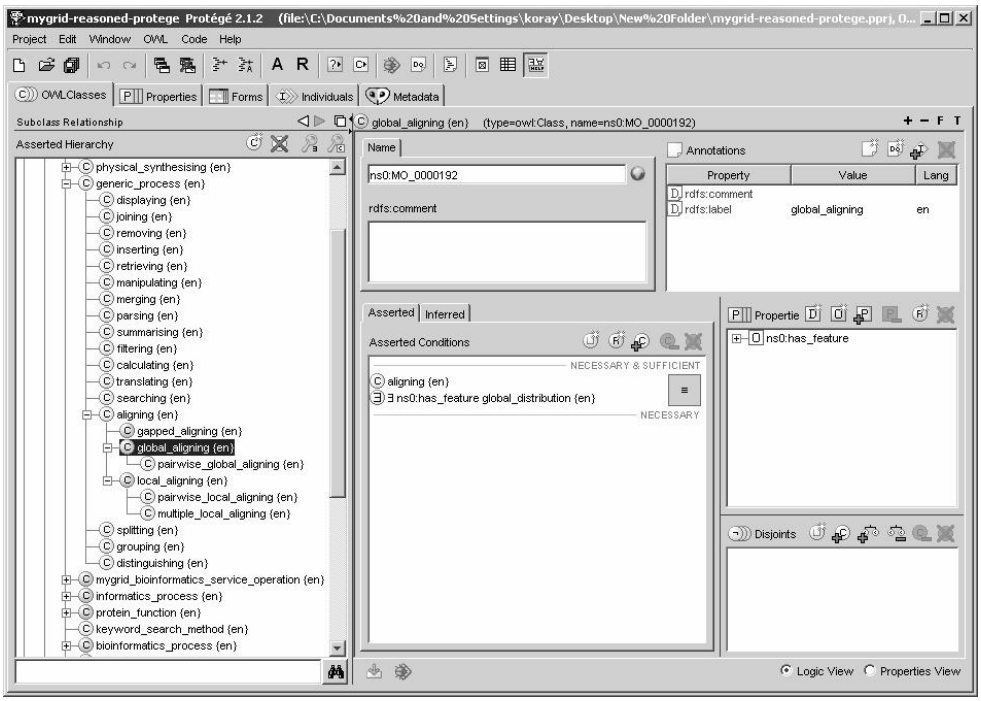


Figure 2.16: The Screenshot of the Protégé Ontology Editor

2.5.6 Use of Ontologies and Reasoning

We have described Semantic Web technologies and their mechanisms for representing knowledge in a machine interpretable way for increased automation of

web related activities. Ontologies not only provide the controlled vocabulary to be used in marking up web resources (e.g. html pages, wsdl descriptions), but also inherently provide a machine-interpretable representation of complex taxonomic and logical relations among these vocabulary terms.

Ontologies can be used to make web search engines smarter by going beyond the current keyword-based approach and allow pages to be found that contain syntactically different, but semantically similar terms.

Another very important application area, which is the focus of our attention, is web service discovery. Service ontologies can be used to generate semantically richer service descriptions that would allow a more intelligent service selection process to take place. This semantic discovery process can be called conceptual trading/matchmaking. In conceptual matchmaking both the service advertisements and service requests are defined ontologically and subsumption reasoning is used to find matches between requests and adverts.

Our previous figure (Figure 2.14) of ontology languages also display examples of such service ontologies, namely DAML-S and OWL-S, which describe characterisic aspects of Web Services. DAML-Services (DAML-S) is expressed in DAML+OIL and OWL-S which is an evolved version of DAML-S is expressed in OWL. These ontologies will be analysed in detail in the following sections.

2.6 Semantic Web Services

Characteristics and Stakeholders The holy grail of semantic web services (SWS) is to enable software agents to: discover previously unseen services; compose discovered services to achieve a given goal; and invoke services without any need for human-intervention at any of these stages. Such a level of increased automation would allow software agents to achieve a high level task like "Make my travel arrangements for the next WWW conference" given by a human. To enable software agents to perform these tasks, machine understandable conceptual descriptions of web services are required.

Having said that, we should also underline that within SWS research, the possibility of human-users being involved in service related activities, and particularly discovery, is not ruled out. As we will be surveying in the following sections humans as well as software agents can benefit from a smarter service selection mechanism that is enabled by use of semantic web technologies.

The state of the art on SWS discovery can be analyzed under two categories of approaches which are top-down and bottom-up.

- Top down approaches, based on the research from the Agents community [86] [62][8] and Problem Solving Methods community [17][67], start with developing their own high-level models of web services and provide groundings of their models to existing Web Service technologies such as WSDL or UDDI.
- Bottom-up approaches that (1) adopt simpler semantic service descriptions that do not cover service composition (process) and invocation layers and (2) use simpler reasoning procedures for discovery.

2.6.1 Semantic Web Service Discovery: Top-Down approaches

In the following three sub-sections we will introduce and compare three approaches to service description namely, OWL-Services (OWL-S) [86], Web Services Modelling Framework (WSMF) [17] and Internet Reasoning Service-II (IRS) [67], and present how each approach has been (or is being planned to be) used to realize the SWS vision.

2.6.1.1 OWL-S: OWL Services

Among the three approaches the most popular and adopted one is OWL-S [86]. OWL-S is an upper ontology for services developed by the agent community. OWL-S supplies providers of web services with a core set of markup language constructs for describing various aspects of a web service, including its capabilities, non-functional aspects (e.g. location, industrial classification, quality of service parameters), interaction patterns, and its invocation details in an unambiguous, computer-intepretable form.

OWL-S, which is expressed in the ontology language OWL, is a follow-up of a former effort named DAML-S [11], which was expressed in DAML+OIL.

The Information Model The OWL-S ontology is structured into three sub-ontologies that describe different aspects of a web service. These parts are :

- Service Profile: describes "what the service does" and is aimed to be used during service discovery. It provides the constructs to generate a black-box description of the capability of the service via 'modelling elements'

for Inputs, Outputs, Pre-conditions and Results (also termed as Effects) (IOPEs). The profile ontology also provides constructs for describing non-functional aspects of a service. The sub-ontology explicitly models the properties: service name, textual description, service parameter, and service category. The category and parameter elements are generic properties that can be used to classify services with industrial classifications or to define non-functional aspects of services.

- Service Model: describes "how the service works" and is aimed to be used during interaction with the service. The execution of a service is seen as a process/workflow execution, and constructs to describe the service's constituent steps and their composition are provided. The Service Model ontology provides process constructs (atomic and composite) each of which also has IOPEs. Process constructs are brought together by control constructs, for which there are four types: sequential, concurrent, conditional, and iteration.

- Service Grounding: describes "how to access the service" and is used for executing the service. The constituent steps of the service process model are mapped to concrete services. The process model is mapped to WSDL constructs that describe real-world service endpoints. Even though grounding to WSDL is not mandatory, grounding constructs for mapping to descriptions other than WSDL have not been proposed.

The OWL-S profile and process models of our example BLAST sequence alignment service can be seen in Figure 2.17 and Figure 2.18 respectively.

The profile description (Figure 2.17) for the BLAST service provides service's geographic location and quality rating as service parameters. Service parameters are the metadata constructs of the OWL-S Profile ontology. The service's inputs and outputs are just identified by their URI's which point to their description in the Service Process ontology.

The process ontology for our example service (Figure 2.18) defines a single atomic process named *doBLAST* for the service. Please note that the semantic types of inputs and outputs of the service are defined by associating them with corresponding concepts from the domain ontology (defining them as instances of concepts from the ontology). In our example this domain ontology is the ^{my}Grid domain ontology which defines the semantic types of inputs to be *SWISSPROT* ,

BLAST and *DNA_sequence* and the outputs to be *BLAST_Report*. Our example does not have pre-condition or post-condition descriptions. However OWL-S's process ontology allows specification of pre/post-conditions as axioms written in an ontology language or rules written in a Rule language.

OWL-S Based Discovery Systems. From the discovery perpective the significant part of OWL-S/DAML-S ontologies is the Profile ontology, which has received the most attention from the pioneer adopters of SWS technology. OWL-S based service Traders commonly use DL reasoning and sometimes additional Information Retrieval Techniques to enable semantic service search.

In [73] a UDDI registry has been augmented with a DAML-S based matchmaker that stores links to DAML-S profile descriptions for services in the UDDI registry. The matchmaker is augmented with a DL reasoner that accepts service search requests which are also represented as DAML-S service profiles. By performing DL reasoning the matchmaker evaluates matches between service advertisments and service search requests. The matches can be of varying degree. The possible scenarios during service search are exact match, inexact match, plug-in match and failure. An exact match between service request R and advertisement A happens when A and R are the same conceptual descriptions. A plug-in match happens when the Advertisement subsumes the Request, a subsumes match happens when the request subsumes the advertisement, and finally a failure occurs if there are no subsumption relationships between conceptual descriptions of adverts and requests.

Similarly in [56] DAML-S profiles are used for representing service adverts and services requests, and a DL Reasoner is used to develop a semantic matchmaker that extends the previous matchmaker's algorithm [73] to add another degree of matching, intersection match, to the service selection process. An intersection match occurs when the intersection of the conceptual descriptions of Requests and Adverts lead to a new conceptual definition which is satisfiable (i.e. non-contradictory). This matchmaker has been built to demonstrate the practicality of the use of a subsumption reasoner during service discovery. The system shows that DL reasoning technology can cope with the scalability requirements of service discovery in a realistic e-commerce scenario.

In [49] a well-known algorithm for matchmaking among agents named LARKS [84] is used within a semantic service matchmaker integrated with a UDDI registry. The notable difference of this matchmaker from the others is that the

```
...

<profile:serviceName>BLAST Service</profile:serviceName>
<profile:textDescription>

          Basic Local Alignment Service

</profile:textDescription>


<!-- specification of geographic location -->

<profile:serviceParameter>
  <addParam:Location rdf:ID="BLAST-Location">
    <profile:serviceParameterName>
        BLAST Service Geographic Location
    </profile:serviceParameterName>
    <profile:sParameter
              rdf:resource="&countryOntology;#UnitedKingdom"/>
  </addParam:Location>
</profile:serviceParameter>


<!-- specification of quality rating for profile -->
<profile:serviceParameter>
  <addParam:QualityRating rdf:ID="Rating">
    <profile:serviceParameterName>
        ServiceQualityRating
    </profile:serviceParameterName>
    <profile:sParameter
              rdf:resource="&ratingOntology;#Good"/>
  </addParam:QualityRating>
</profile:serviceParameter>


<!-- specification of quality rating for profile -->

<!-- Descriptions of Inputs Outputs -->

<profile:hasInput rdf:resource=
                   "&blastProcessOntology;#program"/>
<profile:hasInput rdf:resource=
                   "&blastProcessOntology;#database"/>
<profile:hasInput rdf:resource=
                   "&blastProcessOntology;#query"/>
<profile:hasOutput rdf:resource=
                   "&blastProcessOntology;#result"/>

...
```

Figure 2.17: The OWL-S Profile of a Sequence Alignment Service

```
....

<process:AtomicProcess rdf:ID="doBLAST">

  <process:hasInput rdf:resource="#program"/>
  <process:hasInput rdf:resource="#database"/>
  <process:hasInput rdf:resource="#query"/>

<process:hasOutput rdf:resource="#result"/>

</process:AtomicProcess>


<process:Input rdf:ID="database">
<process:parameterType rdf:datatype="&xsd;#anyURI">

&myGrid_Domain;#SWISS_PROT

</process:parameterType>
</process:Input>

<process:Input rdf:ID="query">
<process:parameterType rdf:datatype="&xsd;#anyURI">

&myGrid_Domain;#DNA_sequence

</process:parameterType>
</process:Input>

<process:Input rdf:ID="program">
<process:parameterType rdf:datatype="&xsd;#anyURI">

&myGrid_Domain;#BLAST

</process:parameterType>
</process:Input>

<process:Output rdf:ID="Result">
<process:parameterType rdf:datatype="&xsd;#anyURI">

&myGridDomain;#BLAST_REPORT

</process:parameterType>
</process:Output>

...
```

Figure 2.18: The OWL-S Process of a Sequence Alignment Service

layering of the OWL-S ontology suite is collapsed into one single representation for services where there is a single profile like description that directly maps to a WSDL operation at the grounding level. In addition to DAML-S's association of ontology terms with service inputs/outputs the system allows further constraints to be defined on these using rule languages. During service selection a set of filters are applied. These filters can selectively be turned on and off to tune speed and accuracy of service search. Examples of speed enhancing relaxed match filters are the description namespace consistency filter that checks whether adverts and requests use same vocabulary, the text filter that calculating similarity between textual descriptions within requests and adverts based on term frequency and so forth. Exact matching filters are those that operate over the reduced space, such as the subsumption and rule evaluation filters. Details of the algorithm can be found in [49].

2.6.1.2 IRS-II: Internet Reasoning Service

IRS-II [67] has a knowledge-based approach to Semantic Web Services that has roots in research on reusable software components. IRS-II is based on the Unified Problem-solving Method description Language (UPML) [31] developed within the IBROW Project.

Information Model UPML uses logical formalisms and ontologies to describe the problem solving capabilities of software components. The components of the UPML framework that have given way to the modelling elements of IRS-II can be seen in Figure 2.19. The UPML framework (hence IRS-II) distinguishes between the following classes of components (each corresponding to ontologies) to describe a service's capabilities [67]:

- Task models: they provide a high level generic description of the task to be solved. The Task element is used to model service requestors' search requests in the web services environment. Task models contain descriptions of inputs, outputs, the goal to be achieved and applicable preconditions. The Task description for our example BLAST Service is given in Figure 2.20. Similar to the OWL-S Profile ontology, the task description provides information on inputs, outputs and ontological terms that define their domain type.

- Problems solving methods (PSM) provide implementation independent descriptions of reasoning processes that can be used to solve a task. PSMs can be seen as abstract algorithms for achieving solutions to stereotypical knowledge-intensive tasks (e.g. diagnosis, classification, design, monitoring, etc.). They act as reasoning templates that need to be instantiated with domain knowledge for each new application (e.g. medical diagnosis, mechanical vehicle diagnosis). Similar to tasks they can have pre/post conditions, inputs and outputs. The PSM definition for our example service is given in Figure 2.21. It has a very similar structure to the task. The only difference is that it provides an additional pre-condition stating that the database needs to exist in order for the service to execute.

- Domain Models which contain the necessary knowledge to instantiate the reasoning templates for different domains.

- Bridges that provide mappings between the above three elements. An example of a bridge would be an ontology that is used to map between two different ontologies used to represent tasks and PSMs.

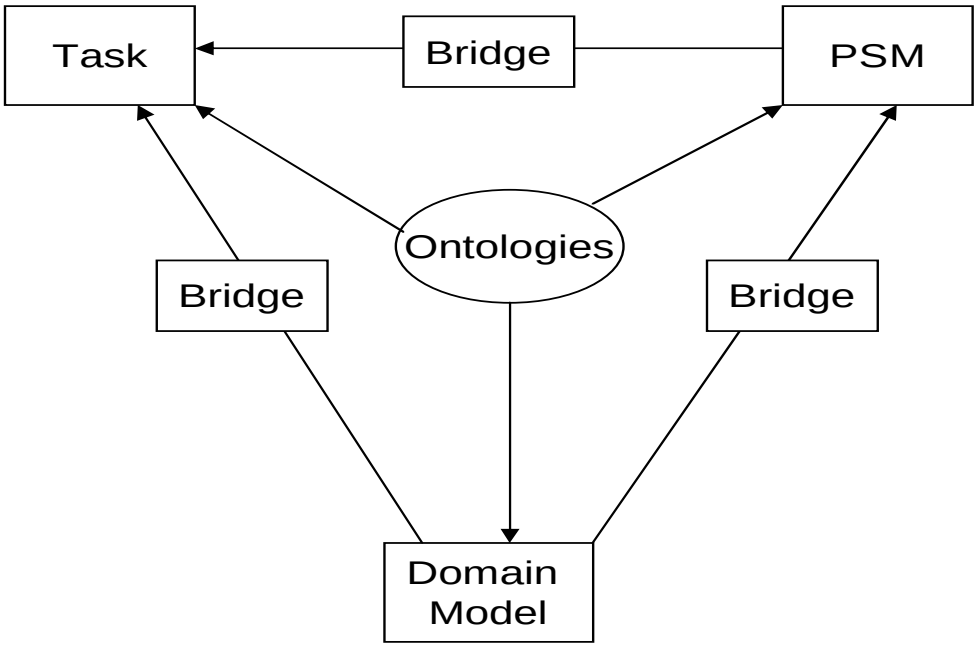


Figure 2.19: The Unified Problem-solving Method description Language (UPML) Framework.

```
(def-class alignment_report_provision (goal-specification-
task)
  ?task
    (( has-input-role    :value database
                         :value program
                         :value query)

     ( has-output-role   :value report)

     ( database  :type SWISS_PROT)
     ( program   :type BLAST)
     ( query     :type DNA_sequence)
  (has-goal-expression
     :value (kappa  (?psm ?sol)
                    (= ?sol  (BLAST_REPORT
                             (role-value ?psm database)
                             (role-value ?psm program)
                             (role-value ?psm query)))))))
```

Figure 2.20: A IRS-II Task Description in OCML.

Within IRS-II all modelling elements of UPML are represented in an ontology language called the Operational Conceptual Modelling Language (OCML). The OCML modelling allows specification and operationalization of functions, relations, classes, instances and rules. By operationalization it is meant that descriptions in OCML can be interpreted and used as working code.

IRS-II uses PSM and Task elements of UPML to describe web services (e.g. a Sequence Alignment Service) and the goals that can be achieved by them (e.g. obtaining a BLAST Report) respectively. By distinguishing between tasks (i.e. service requests that can be made) and PSMs (i.e. methods to achieve the requested tasks) IRS-II allows a service to be associated with multiple tasks and a task with multiple services. An example of such a case could be given from the Amazon web services [1]. These services might both be used for bibliography search and book buying. Using UPML's separation of tasks and PSMs it is possible to advertise these two capabilities for the same service.

The UPML Framework allows PSMs (i.e. services) to decompose tasks into subtasks for their solution. However, this aspect of the framework has not been reflected within the IRS-II implementations, therefore each PSM corresponds to an atomic service in IRS-II.

[1] `http://www.amazon.com/gp/aws/landing.html`

```
(def-class sequence_alignment_service (primitive-method)
  ?psm
    (( has-input-role    :value database
                         :value program
                         :value query)

     ( has-output-role   :value report)

     ( database  :type SWISS_PROT)
     ( program   :type BLAST)
     ( query     :type DNA_sequence)
     (has-precondition
        :value (kappa  (?psm) (database-exists
                                 (role-value ?psm database))
     (has-postcondition
        :value (kappa  (?psm ?sol)
                       (= ?sol (BLAST_REPORT
                               (role-value ?psm database)
                               (role-value ?psm program)
                               (role-value ?psm query)))))))
```

Figure 2.21: A IRS-II Problem Solving Method Description in OCML.

Similar to OWL-S's grounding ontology IRS-II also allows description of binding information for inputs and outputs of PSMs. The parameter binding information together with an actual list of providers (i.e. endpoints) for a certain PSM is kept within the discovery system.

Discovery Mechanisms In addition to its UPML based Information Model, IRS-II also provides an implemented infrastructure that embodies a service selection component called the IRS-Broker.

Within IRS, tasks and PSMs are linked to each other by a "tackles-task" relation which connects a PSM to a Task. For our example the relation is *(tackles-task sequence-alignment-psm sequence-alignment-task)*. This association is based on the assumption that PSMs are always created in the context of some task that they solve. When a request arrives to achieve-a-goal IRS-II finds all the PSMs which are linked to the specified task via a tackles-task relationship.

In short, even though PSMs and tasks are semantically described by associating ontology terms with their inputs, outputs and pre/post conditions these descriptions are not exploited in any way during service selection. This is due to the assumptions on (1) the tackles-task relationship guaranteeing that a PSM actually addresses a task and (2) service requestors have discovered their goal (i.e. task) descriptions. This is quite a limiting assumption when we compare

IRS-II to all other service discovery systems; in fact we can claim that IRS-II does not perform discovery from our context. Our criticism of IRS-II is justified by recent efforts for a Task/Goal discovery tool to be incorporated into the most version of IRS, IRS-III. The goal discovery tool allows human-users to find goals based on their inputs/outputs, pre/post conditions.

2.6.1.3 WSMF: Web Service Modelling Framework

WSMF [17] is an e-commerce oriented SWS effort, which also has its roots in UPML. WSMF aims to exploit semantic descriptions and reasoning facilities at all stages of a real life e-commerce scenario. Similar to OWL-S, WSMF provides an ontology called the Web Service Modelling Ontology (WSMO) to enable semantic web service discovery, composition and execution.

Information Model In its current status WSMF is subject to intense development and all of its specifications are in draft status. However the high level conceptual model has been developed [55]. The main modelling components of the WSMO ontology are Goals, Web Services, Ontologies and Mediators.

- Goals specify objectives that requestors of a web service may have. Within WSMO Goals are defined by post-conditions and effects both of which correspond to logical axioms.
- Mediators provide links between modelling constructs. Mediators of WSMO are similar to Bridges in UPML framework.
- Web Services provide descriptions of the following aspects of a service:
    1. Capability: pre-conditions (inputs), assumptions, post-conditions (outputs), and effects of a service by associating each with a logical axiom.
    2. Non-functional properties: a group of pre-defined properties that give metadata regarding quality of service aspects of a web service (e.g. reliability, security, accuracy).
    3. Interfaces: The interface sub-part of WSMO is similar to the OWL-S process model. The Interface ontology is currently under development. However WSMO has identified the following modelling elements to be used for the interface description of web services orchestration: errors, compensation and message exchange patterns, and will provide ontologies for all. Developers of WSMF have criticized OWL-S for not differentiating between inner working mechanisms of a composite web service, and its external interface. Therefore WSMF promises such a

differentiation in its Interface ontology [55]. WSMF architects plan to use a Web services orchestration standard, namely BPEL4WS, which makes such a distinction for composite web services, as a basis for the development of their future Interface ontology

4. Grounding: in order to develop a grounding sub-ontology WSMF plans to make use of the OWL-S grounding ontology.

- Ontologies will be used for modelling knowledge of different domains. They provide the basic glue for semantic interoperability and are used by the three other WSMO components

All modelling elements described above have a fixed set of properties for specifying their cross-domain properties. This common set of properties is incorporated into WSMO from the Dublin Core Metadata Element Set [5]. Examples of these core properties are title, creator, author, subject, description, date, etc.

The formal grounding of WSMF conceptual elements is done in the Web Services Modelling Language. WSML is a modular, frame-based family of formal representation languages with its roots in Description Logics, First-Order Logic and Rule Languages. We will not describe the WSML languages in detail because they are currently subject to development.

Discovery Mechanism

Due to the incomplete status of WSML languages, and WSMO modelling elements, there is no usable WSMO based Trader implementation available. However, developers of WSMO ontologies have reported experiments on the use of the ontology language F-Logic to represent goals and services and use of reasoning to perform service matchmaking. [54].

2.6.2 Semantic Web Service Discovery: Bottom-Up Approaches

In addition to the top-down SWS frameworks — OWL-S WSMO, and IRS that aim to address all activities in the services oriented setting — some research groups have taken more pragmatic approaches using simpler information models for describing services and using simpler reasoning mechanisms for web service discovery.

Such a system that has also managed to reach to the level of practical use is reported in [59]. The Trader system named MOBY-Central is a centralized

registry of biological web services. The information model supported registry has an atomic view of web services (i.e. single operation). A service is defined as a combination of input/output parameters, their object types, and the service with its service type. Hierarchies of biological Object-Types and Service Types are stored within MOBY-Central. These hierarchies can be seen as simplified biological domain ontologies for data and services. During service search, which can be done via the MOBY-Central API, it is possible to search for services based on their service types and the object types of their inputs/outputs. The registry also takes the sub-class relationships into account during service search.

In [91] a framework, named METEOR, for semantically organized peers of service registries is described. Similar to MOBY-Central, this system provides semantic discovery by allowing parts of WSDL files, namely the operations and message part elements, to be annotated by ontology terms. The system has simplified ontologies in the form of taxonomies of terms for describing (1) semantic types for data that can be input and output to services, and (2) semantic types for services representing their functionalities (e.g. flight_reservation_service). These taxonomies are stored in UDDI. The system allows clients to search for services based on the semantic types of their inputs, outputs and functionality by taking the hierarchical relationship of ontology terms into account.

2.6.3 Remarks on Semantic Web Services

The application of Semantic Web technologies to services is mainly aimed at enabling software agents to perform service discovery, composition and invocation on behalf of humans. Therefore three conceptual SWS frameworks OWL, WSMO, and IRS try to address all these activities, by providing machine-understandable specifications for service capability description, service composition description, and service execution mechanisms (see Semantic Web Services columns of Table 2.1). From the discovery perspective the significant part is the capability description and non-functional properties of services.

At the capability description layer OWL-S provides the single layer Profile sub-ontology, which makes no distinction between the task performed by a service and the actual service description. In contrast IRS-II and WSMF make such a distinction by providing two layers (i.e. Task/PSM, Goal/Web Service layers) of modelling elements for capabilities. The reason for keeping tasks separated from the actual web service descriptions is a possible n-to-m mapping between them,

i.e. the same web service can serve different tasks and different (competing) services can serve the same task. Such a separation also allows usage of different terminologies in tasks and service descriptions. However, when tasks/goals and web services are separated, as in WSMF and IRS-II, it is necessary for the tasks/goals (possible service requests) to exist prior to service publishing/discovery. Services can only be advertised linking them to an existing task/goal. Similarly, during discovery requestors need to point out a goal that needs to be achieved by the service. However, it has not been addressed by WSMF or IRS-II how the requestors find or define their goals.

On the other hand the bottom-up approaches also provide a service capability description either by using existing web service description technologies (WSDL) or by designing their own simple models.

In both approaches the service selection procedure involves inferencing mechanisms ranging from simple (crawling of subclass hierarchies) to complex (description logic reasoning between conceptual definitions) to match service advertisements with service requests.

SWS is quite a new area of research and most of the semantically enhanced Traders developed in this area are not in widespread use. Therefore, up to now, issues that could arise upon deployment and wide spread use have not been the focal point of SWS research agenda. However there are early reports of experiments in the performance area. In [48] a performance analysis for the previously described (see Section 2.6.1.1.) semantically enhanced Trader has been given. It is reported that the Trader is publicly deployed and has proved to be *"practically usable"* [48] with certain assumptions such as putting an upper limit on registered services and excluding ontology loading time. Regarding reliability of content and timeliness of service information in SWS Traders, there is no known research experiment. This is not surprising since conventional Web Service Traders (without semantics support) suffer from unreliability and staleness of content [21] and this aspect has recently started gaining research attention in the web services community [9]. As may be recalled from the Grid or the Distributed Object Traders, issues such as scalability and reliable Trader content were being addressed by means of soft-state registration and Trader federation.

2.7 Semantic Grid

Characteristics The final area of distributed computing from which we will analyze a Trader is the Semantic Grid [40]. With an influence from the Semantic Web activity, the Semantic Grid is defined as [2] *"an extension of the current Grid in which information and services are given well-defined meaning, better enabling computers and people to work in cooperation"*. We interpret the term *services* as referring to the diverse types of computational resources and services that provide access to them on the Grid (both first and second generation).

The Semantic Grid activity is seen as an important enabler for the current e-Science activity which is defined as *"global collaboration in key areas of science and the next generation of infrastructure that will enable it"* [30]. There are a large number of projects currently undertaken, especially in the UK to realize the e-Science vision. The myGrid project in the context of which our work falls is also a part of the e-Science activity.

From a discovery perspective, efforts in myGrid can be classified in the SWS area rather than the Semantic Grid area. This is due to myGrid's focus on discovery of information producing and consuming service entities rather than discovery of different computational resources. However, at the same time services in myGrid have certain characteritics which coarsely describe the resources they are associated with (recall the **resource** and **application** characteristics of myGrid service entities). OGSA based Second Generation Grid defines how the future Grid infrastructure will be. Therefore efforts on semantic service discovery in myGrid can also be seen as early experiments of describing services and their associated resources on the Bioinformatics Grid.

Currently there are no examples of a semantically enabled second generation Grid Trader whose information model fully supports the OGSA vision and its enabling infrastructure, the Web Services Resource Framework (WSRF). This is mainly due to exploratory research efforts on deciding how the service oriented Grid should be realized. For the time being the specifications seem to have stabled with the recent WSRF effort. Hence we anticipate that there will be more efforts on semantically describing and discovering services conforming to the WSRF framework in the near future.

On the other hand, in parallel to efforts in myGrid, there are pioneer examples

[2] http://www.semanticgrid.org/

of efforts on integrating semantic web technologies for discovery on non-service based Grids [85] [24]. Within this section we will analyze one of these semantic Grid Trader systems built within the Geodise e-Science project.

2.7.1 Geodise

The Geodise [24] toolkit is a suite of tools for Grid-enabled Computational Fluid Dynamics analysis, design optimization and search within the Matlab environment. The Geodise toolkit acts as a client to remote compute resources that are exposed to the Grid as Matlab Functions. These functions are grouped in three categories as: (1) functions which allow the user to run and control jobs on Grid compute resources, (2) functions which are used to archive, query, and retrieve data, and (3) functions which are used to notify the user. These functions are brought together in workflows within a knowledge based problem solving environment.

Information Model. Geodise Functions are described in the Engineering Design Search and Optimisation (EDSO) Tasks ontology. A screenshot of the OilEd ontology editor displaying the EDSO ontology is given in Figure 2.22. EDSO function ontology is a DAML-S influenced ontology written in DAML+OIL for describing Matlab functions in terms of their inputs, outputs, the task they perform as classification functionalities based on the aforementioned three categories, the tool set they invoke, or the algorithm they implement.

Discovery Mechanism. To take full advantage of the function descriptions expressed in DAML+OIL, Geodise makes use of DL reasoning during function discovery.

The function trader of Geodise is a knowledge-based system called the Instance Store [45]. The Instance Store provides efficient DL reasoning by the help of database querying. The Instance Store infers the classification hierarchy among concepts in an ontology and then stores this classification and the instances (a.k.a. individuals) of concepts in the classification in a database. Within Geodise, conceptual descriptions of Matlab functions are stored in the Instance Store.

Additionally, the Geodise environment provides requestors with a GUI tool that helps to build a conceptual descriptions of the function that they wish to incorporate into their workflow. This description of a request for a function is then sent to the Instance Store which for conceptual matchmaking between requests and adverts.

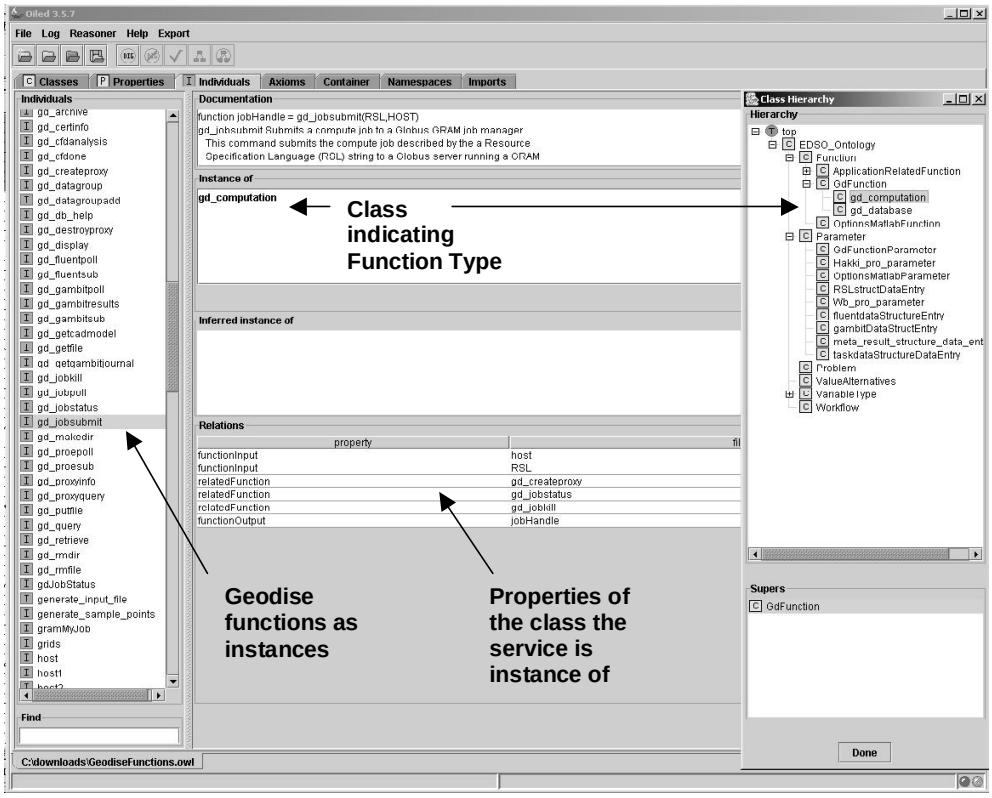


Figure 2.22: The screenshot of EDSO Ontology displayed in the OilEd ontology editor.

There are two types of stake holders who might initiate discovery in Geodise, Firstly, the users may search for functions they would like to incorporate into their workflow by making use of the GUI search tool. Secondly, the workflow development environment can use discovery to make suggestions to the user on the functions that might come next within a workflow. Such suggestions are based on matching signatures (inputs/outputs) of services.

2.8 Summary

A comparison of the basic discovery approaches surveyed in this chapter is given in Table 2.1.

Based on the remarks that we have made at the end of each section we can draw the following conclusions

- The components subject to discovery on the Grid are described by a single-layered Information Model. This is due to the fact that these components generally exhibit a single capability which is characterized by what the component is rather than what it does. Moreover, in order to describe Grid resources in a way that would enable effective discovery, Traders in the Grid use simple attribute value based information models such as GLUE.

- For the case of distributed objects (DO) discovery and existing web service (WS) discovery technologies there is a great similarity in the way descriptions are made and discovery is provided. This is largely due to the fact that WSs have emerged from DOs and they essentially provide similar functionality from a discovery perspective. Within DO traders objects are typed and discovered by Service Types which is a combination of the interface specification of the object and its properties, whereas in WSs, the typing and discovery is done over the tModel mechanism. Due to the openness of the WS environment the types that tModels correspond to are not defined within the Trader. Interpretation of typing of services is left to particular domains with different needs; however, typing with respect to the interface description (i.e.WSDL) is suggested. We have previously mentioned that in WS and DO environments the type of entities subject to discovery are of a single kind, a service, or a software object. These entities are

	First Gen. Grid		**Second Gen. Grid**	**Distributed Objects**	**Web Services**		**Semantic Web Services**				**Semantic Grid**
	MDS2	Condor	MDS3	CORBA	UDDI	UDDI Enhancements	OWL-S Based Systems	IRS-II Broker	WSMF	Bottom-Up Appr.	Geodise
Back-End Data Model	LDAP	Condor Class AD language	XML	Irrelevant	Irrelevant	Irrelevant	DAML+OIL or OWL	OCML	F-Logic	Irrelevant	DAML+OIL
Capabilities	GLUE	Schemaless ClassADs	GLUE	Service types	tModels	tModels	Profile	Task PSMs	Goals WS Capabilities	Service Types I/O Types	Geodise Model of Functions
Non-functional Properties				Object Properties	tModels	properties + QoS extensions	Service Profile	—	WS Non-functional properties	—	—
Binding Information				IDL	WSDL	WSDL	WSDL Grounding	SOAP Grounding of Parameters	WS Grounding	WSDL	—
Lookup via Unique Names.	Hierarchical	—	—	Hierarchical	—	—	—	—	—	—	—
Discovery over Inf. Mod.	LDAP Protocol	ClassAD property evaluation and ranking	XPath Queries	OMG Trader API, Discovery by Service Type	UDDI API Discovery by tModel	UDDI API Discovery via tModels and properties	DL Reasoning	—-	F-Logic Reasoning	Discovery via Service Type and I/O types, Simple Reasoning	DL Reasoning
Discovery Purpose	Job Submission			Binding	Composition & Invocation						
Stakeholders initiating discovery	Software Agents and Humans				Humans	Humans	Intelligent SW agents and Humans				
Multi-Trader Support	+	+	+	+	+	-	-	-	-	-	-
Temp. Registration	+	+	+	-	-	+	-	-	-	-	-

Table 2.1: Comparison of Discovery Systems Surveyed

typically characterized with their inputs and outputs and they exhibit multiple capabilities in different domains. However, the WS and DO modelling approaches and their Traders provide no means for describing the actual interpretations of capabilities of services by means of actual interpretations of whet input/outputs actually are. This deficiency does not cause much problem given that they operate in a closed environment; however within an open environment such information is crucial.

- The SWS effort aims to tackle this problem by incorporating formal semantic into descriptions of services and pointing out the need of domain specific knowledge. The grey-shaded area in Table 2.1 denotes that the modelling frameworks of SWS need to be augmented with knowledge of different domains to be used for discovery. The formal approach to richer descriptions also enable increased automation via machine understandability. Within all SWS approaches we observe that descriptions are made in knowledge representation formalisms with different levels of expressivity. And we also observe that the Trader undertakes its job by use of inferencing procedures to provide smarter service selection given the formal and knowledge rich service descriptions. Among the SWS efforts we see WSMO and OWL-S as the candidates most likely to become the standards in SWS in the future. We would like to emphasize that WSMF provides larger coverage of aspects of a service, in terms of its non-functional properties which we think will be of great importance when SWS become mundane. Furthermore, we also anticipate that the issues related to scalable discovery and reliable (up to date) Trader content will be of great importance once deployment of these technologies happen.

- The bottom-up discovery approaches in SWS have simpler information models and simpler reasoning mechanisms when compared to top-down approaches. These systems are focused on providing descriptions only at the capability layer in terms of semantic typing of service functionalities, and inputs/outputs of services. Their discovery mechanisms employ simple forms of reasoning where sub-class or is-a relationships among domain terms used in descriptions are taken in account.

- In the Semantic Grid, we observe that the SWS models may not be appropriate or may need extending to describe diverse types of resources subject

to discovery. For the particular case we analyzed, namely Geodise, resources were Matlab functions with custom functionalities, and they were characterized not only by their by their inputs/outputs and function types but also by the resources they access, the toolset they invoke, etc.

As stated in Chapter 1 the resources subject to discovery in myGrid are information consuming and producing entities, a subset of which are web services. As a consequence, the service discovery setting in myGrid contains capability description models and service search mechanisms similar to those in SWS. On the other hand, together with other e-Science projects myGrid is a pioneer for the Semantic Grid. In fact the myGrid model of services have a lot of similarities with the EDSO model of Matlab functions in Geodise. In both of these projects the entities are characterized by inputs/iutputs and a characterization of their functionality, the resources they access, the algorithms they use, or the tool set they belong to. This similarity demonstrates convergence of efforts for not only describing services but also describing the resources that are accessible them which will be widely used in the next generation Grid infrastructure.

Chapter 3

Service Discovery in myGrid: Early Efforts

This chapter aims to elaborate the service discovery setting in myGrid. In order to meet some of the requirements for service discovery outlined at the beginning of this thesis, myGrid has previously experimented with two solutions that could be categorized within the Semantic Web Services area. These two solutions, which we refer to as the 'Semantic-Rich' Approach and the 'View-Only' Approach, have explored the boundaries of the research questions outlined in Chapter 1. Both approaches have implemented the Provider-Trader-Requestor architecture. Furthermore, both approaches have reflected to the requestors an information model, that captures their view of different types of entities subject to discovery. This information model is called the myGrid Service Schema V1. (see gray-shaded hexagonal in Figure 3.1). The differences in the two approaches can be summarized as follows.

- The **Semantic-Rich Approach** has been given this name due to its adoption of high expressivity in service descriptions via DL based representations. Additionally, DL reasoning has been used at all stages of the discovery lifecycle including the generation of service descriptions and querying of them. In this approach two Traders: one responsible for domain specific searches and the other for domain independent, have worked cooperatively. The Information Model supported by these is a combination of the UDDI Information Model and the myGrid service schema v1 seen in Figure 3.1
- The **View-Only Approach** as its name implies is entirely based on the myGrid component, named the *View*, as the Trader. The distinguishing

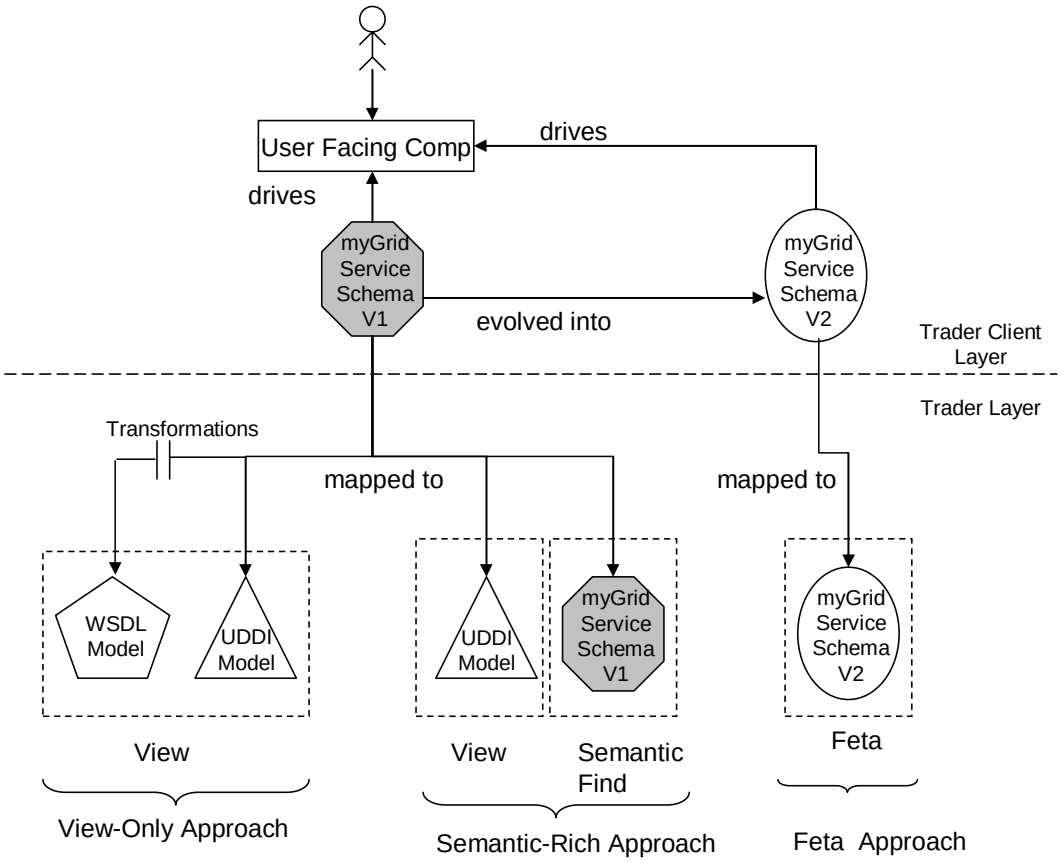


Figure 3.1: A Contextual Diagram of Information Models supported by User-Facing components and the Traders in myGrid Service Discovery Frameworks

characteristic of this approach is its semantic support at the minimum scale, where the the domain ontology has been used as a controlled vocabulary and has been integrated to service descriptions through metadata attachments. No reasoning support has been provided during the generation of descriptions or their querying. Furthermore the Information Model supported by the Trader in this approach has been a UDDI and WSDL based one for describing services and their capabilities. Adoption of WSDL rather than the myGrid Service Schema for service capability description within the Trader has mandated some transformations to take place (see Figure 3.1), which will be described later in the chapter.

These two frameworks have been built by making use of a set of common components to different extents. We will first provide an overview of these common components namely the myGrid Domain Ontology, the Pedro data capture

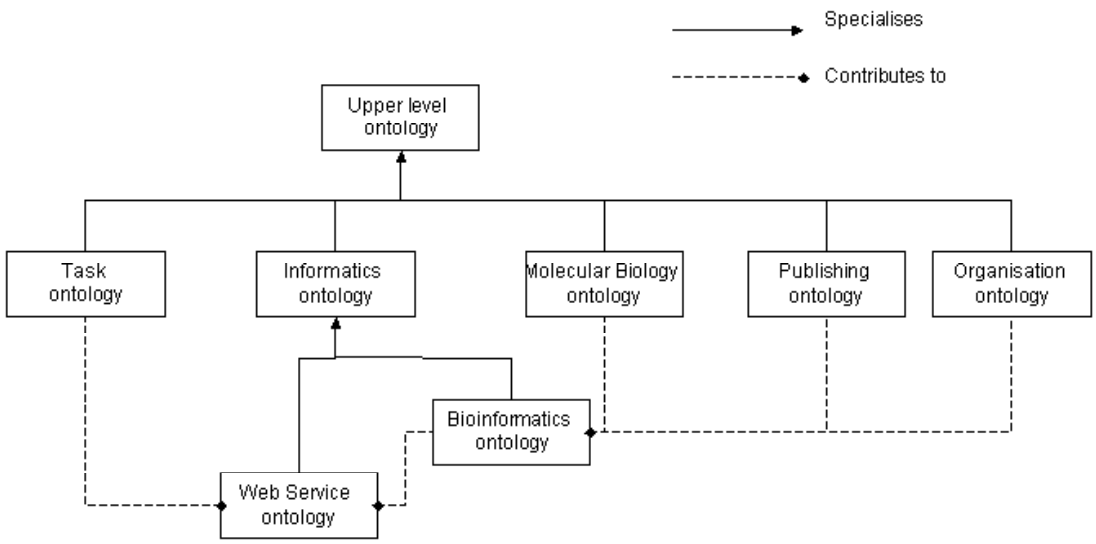


Figure 3.2: myGrid's Suite of Ontologies. Figure is taken from [95].

tool, and the myGrid View service registry. Later the way these components have been used in two approaches will be analyzed in detail, finally we will provide a comparative analysis.

3.1 myGrid Domain Ontology and myGrid Information Model of Services

Within myGrid a suite of ontologies have been developed. These ontologies not only model the knowledge of the domain but also provide the necessary modelling elements for describing bioinformatics resources (databases, tools) and the services that provide access to them.

Figure 3.2 shows the suite of myGrid domain ontologies [95]:

- The upper level ontology is a foundation for all other ontologies; it provides the high-level categories that can be commonly found in a life-sciences ontology, such as Structure and Substance.
- The Informatics ontology captures basic informatics concepts such as data, database, metadata and so forth.
- The Bioinformatics Ontology builds on the Informatics ontology which has a rather generic view and introduces bioinformatics specific resource descriptions such as the SWISS-PROT database, BLAST_Application or EMBOSS

Tool suite

- The Molecular Biology ontology describes molecular biology concepts, data of which is largely subject to processing and integration within the myGrid domain. Examples of concepts in this ontology are protein, nucleic_acid or DNA_sequence.
- The Publishing Ontology provides the concepts to be used to describe scientific literature, which is important source of biological knowledge in the domain. Examples include article, abstract, citation, reference.
- The Organization Ontology provides concepts to describe organizations and the instances of those concepts such as European Bioinformatics Institute.
- The Task Ontology provides a classification of tasks that can be performed by a bioinformatics service in myGrid's domain. Examples include retrieving, aligning, global aligning, local aligning.
- Finally and most importantly the **Web Service Ontology** provides :
    1. An OWL-S profile influenced generic sub-ontology for services also known as the *myGrid Service Schema*, which provides constructs for describing users's view of capabilities of different bionformatics service entities outlined in Chapter 1. Characterization of capabilities is done via attributes for semantic types of input/output parameters of services. In addition to parameter types, other bioinformatics specific service attributes— task, resource, application, method, that can be associated with services —are defined in this schema. When used in descriptions, these attributes link the description to the annotation vocabulary, which corresponds to named concepts within the molecular biology, informatics and bioinformatics sub-ontologies. A diagram of the simplified view of the myGrid Service Schema Version 1 is given in Figure 3.3. The schema supports describing service entities and also their subcomponents. This feature has been particularly developed to be able to describe workflows and their sub-components. Hence there are two entities in this schema both representing units of functionality, both allowed to have input/output parameters and bioinformatics specific aspects. This schema has later been modified and used in Feta as well (see Figure 3.1). A detailed discussion of important aspects of the recent schema will be given in Chapter 4 where we describe Feta. The differences between the myGrid service ontology and other service

ontologies surveyed in Chapter 2 (e.g. OWL-S, WSMO) are those of omission because the requirements of the domain largely simplifies the expected model of services. The myGrid service ontology does not provide any schema similar to OWL-S process or grounding models. The purpose of such models is to describe (a) how to interact with a service and invoke it or (b) how to compose services into workflows. Given that in myGrid the interactions with services are handled by Freefluo in an invisible way and users want to compose services into workflows manually, the need for such schemas is eliminated. The myGrid service schema only captures capability descriptions; other aspects of services, such as non-functional properties, are not modelled in this schema.

2. A group of conceptual descriptions of commonly used bioinformatics services. These descriptions are defined using the classes and properties from the above generic ontology other the other domain sub-ontologies. Examples of these descriptions such as BLAST service, BLASTp and so forth were given in Figure 2.15 of Chapter 2.

The suite of domain ontologies was initially developed in DAML+OIL language, later exported to OWL. In addition to the OWL based version, two less expressive snapshots of the ontology have also been used for service discovery in myGrid. These are the RDF(S) based domain classification and the myGrid domain vocabulary which is a controlled vocabulary of named concepts. The service sub-ontology has also been recast as an XML Schema (see Figure 3.3) and used to configure an XML data entry tool for service annotation purposes.

3.2 Pedro Data Capture Tool

Pedro [37] is an open source tool for ontology aware data entry. Pedro provides a GUI based interface to allow users to generate XML instance documents with respect to a given XML Schema in a simple form filling fashion. Figure 3.4 displays a sample screenshot of Pedro. Panel A shows a hierarchical view of elements in the generated XML instance, Panel B shows the immediate child elements of the selected node in the hierarchy. In Figure 3.4, Panel A shows the elements Operations, OperationInputs, OperationOutputs that exist in the description of the ServiceDescription element named *AffymetrixMapperService*. Panel B shows child nodes of the serviceOperation element named *getSequence*.

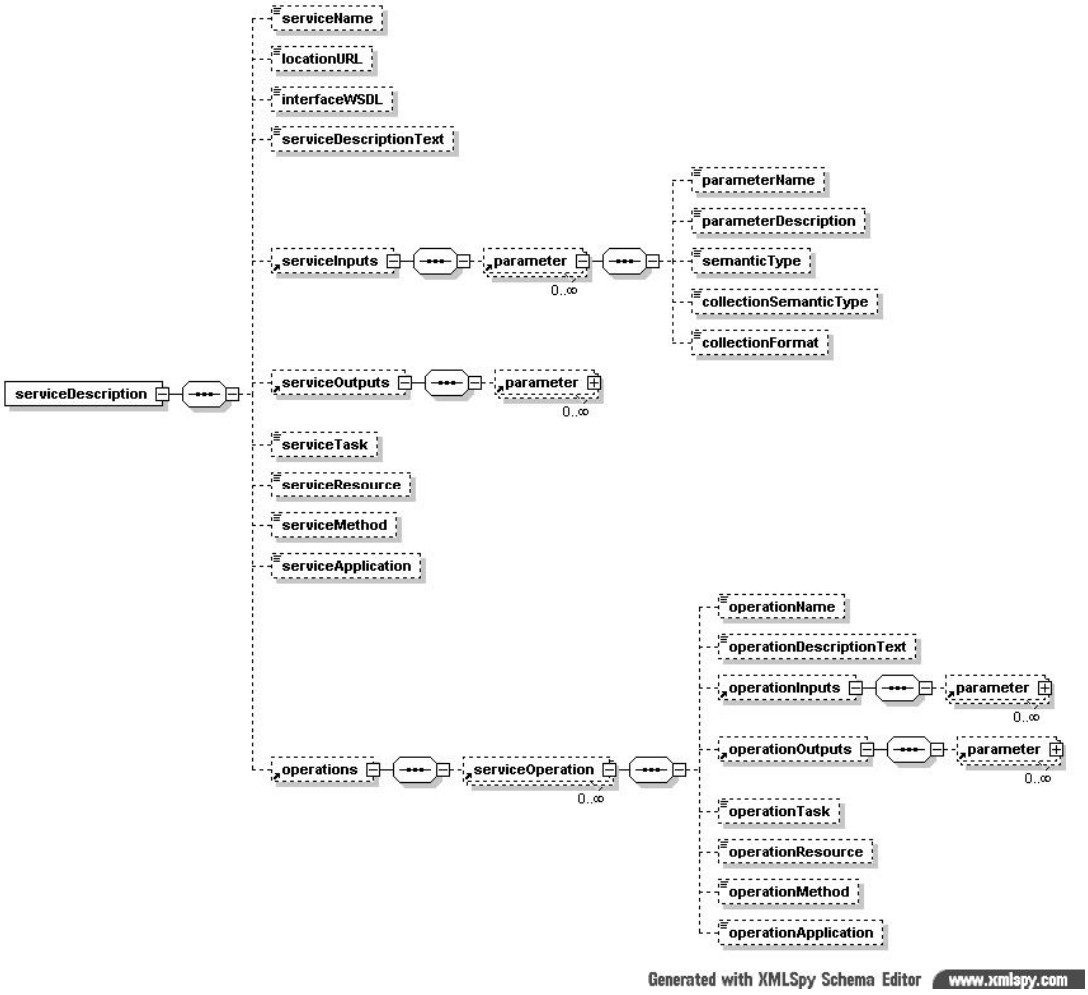


Figure 3.3: The myGrid Service Schema V.1.

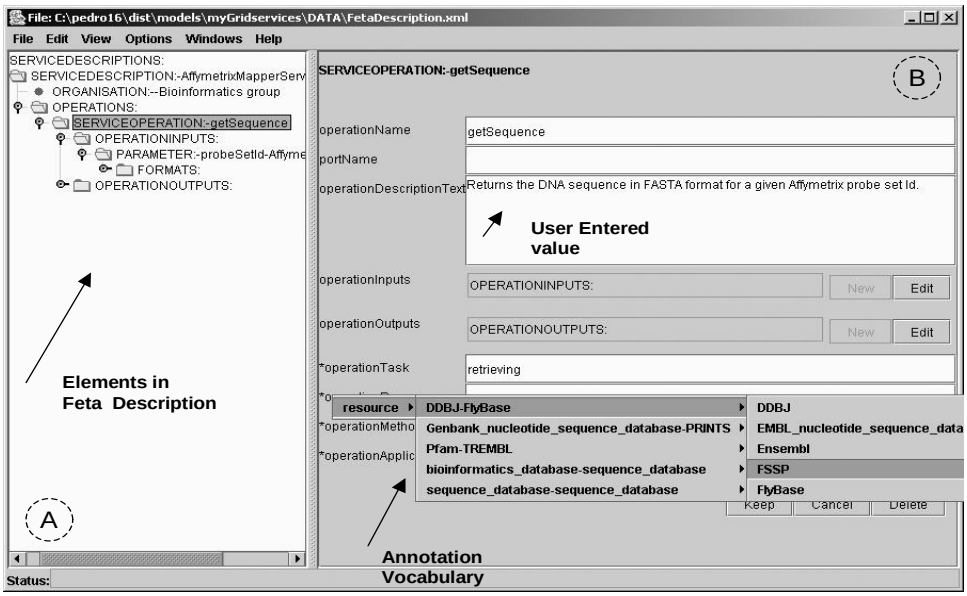


Figure 3.4: A Screenshot of the XML Data Entry Tool Pedro

With its intrinsic support for ontologies, Pedro allows incorporation of ontology terms (i.e. named concepts in the ontology) as XML element values into the generated XML instances. The ontology terms are presented to the users with respect to their hierarchical is-a, or sub-class relationships (see popup menu). These terms are then selected and used as values for particular fields. The *operationResource* field in the screenshot is one such field being filled with term *FSSP*. Incorporation of ontological terms is called the annotation process.

Pedro also allows programmatic access to its inner model of XML instances, which enables getting a hold of the generated document instance and its contents prior to its serialization to XML. This way it is possible to serialize the description to other formats such as OWL. Additionally Pedro provides extensibility support for development of custom form field validation components and custom ontology term supplier components.

3.2.1 Use Of Pedro In myGrid

Pedro has been used to generate domain-specific descriptions of services with respect to myGrid Service Schema. The reasons for using Pedro as the annotator component in myGrid discovery approaches are :

- Pedro is an external tool, which was available prior to development of

myGrid discovery solutions.

- Pedro generates user friendly and customizable data entry forms which fits with the domain's requirements on avoiding necessity for highly skilled service annotators.
- Alternative data capturing environments, such as Protégé's dynamic form generation tool have been found to be less user friendly than Pedro due to strong ties to underlying logical formalisms.
- Necessary human resources for the development of a custom built annotator have not been available during the course of the project.

3.3 myGrid View

The myGrid View [65] [64] is a UDDI-compatible enhanced service registry that supports:

- Federation of contents of multiple registries into personalized registries, named Views. Personalization of federated descriptions is done by attaching metadata to them. The View has an underlying back-end model based on the W3C standard, Resource Description Framework (RDF)[58] which is also the enabler of its metadata facilities.
- WSDL based description of a services to be stored and queried in addition to UDDI based descriptions.
- Other features such as sending of notifications when service descriptions and service metadata are added or removed and policy-based management of registry contents.

Within the following subsections three sub-parts constituting the View's Information Model is described.

3.3.1 UDDI Compatibility

A subset of the View's information model, and its API is UDDI compliant. The information model, including the UDDI part is mapped to an RDF model for storage and querying. The View represents UDDI information model entities and their relationships as RDF resources and properties. Figure 3.5 displays a simplified view of the RDF data generated behind the scenes after the View is populated through its UDDI publish APIs to register a business service entity.

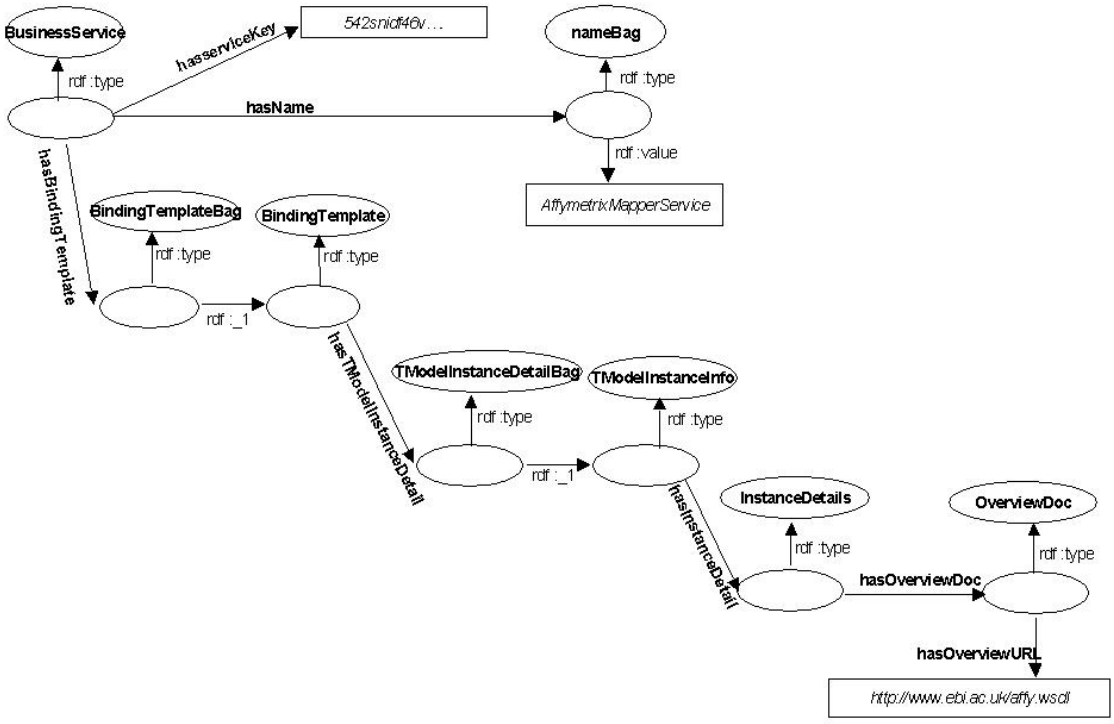


Figure 3.5: RDF data corresponding to UDDI Based Service Information. The unnamed nodes correspond to RDF blank nodes.

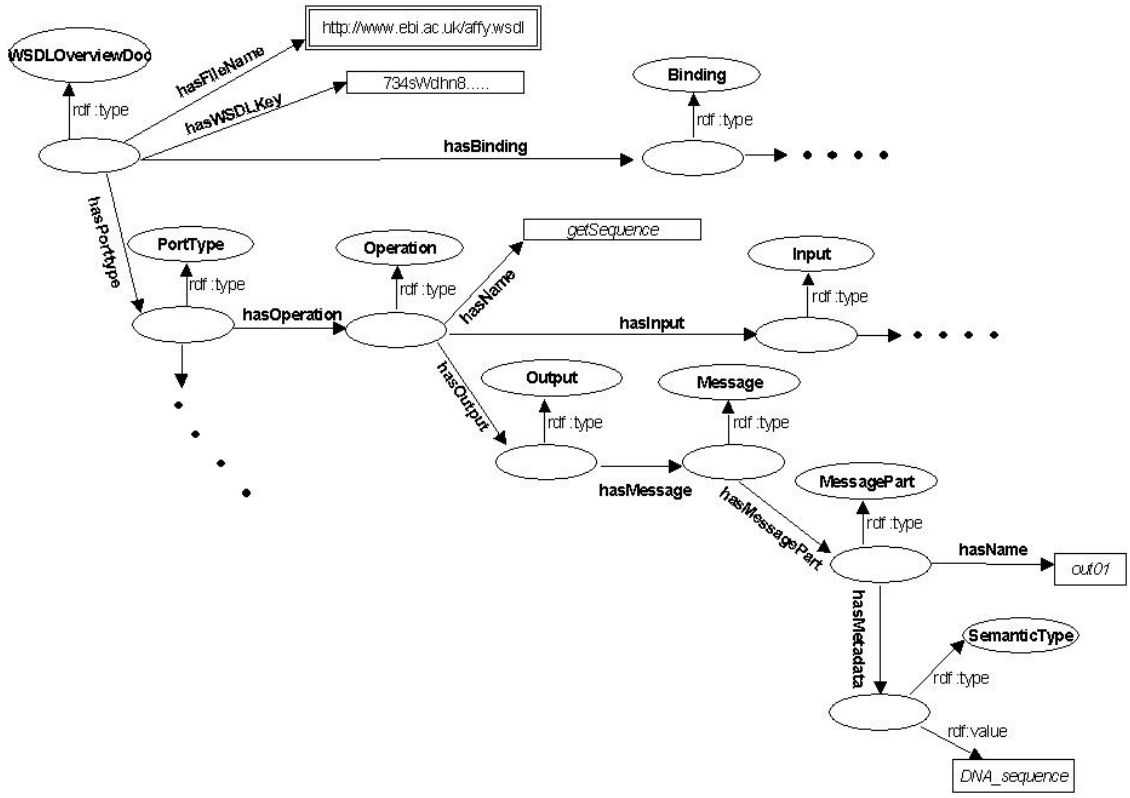


Figure 3.6: RDF data corresponding to a WSDL Description. The unnamed nodes correspond to RDF blank nodes.

3.3.2 WSDL Extensions

The View allows registration of WSDL descriptions for web services. In addition to the registration of links to WSDL descriptions as tModels as in coventional UDDI registries, the View also supports content aware WSDL registration. It processes WSDL descriptions and stores their contents with respect to modelling elements of WSDL such as operation, portType, message Part and so forth. The WSDL extension is based on the assumption that web service capabilities can be described by WSDL descriptions. Figure 3.6 displays the RDF data generated after the registration of a WSDL document for a service. As seen in Figure 3.6 WSDL elements have been mapped to RDF.

While we were introducing WSDL in Chapter 2 we mentioned the lack of semantics in WSDL descriptions. To overcome this limitation, the View allows attachment of metadata to certain parts of a web service description.

3.3.3 Metadata Extensions

The View allows metadata attachments in the form of RDF triples attached to certain elements in the registry information model via a *hasMetadata* link. Examples of these are *BusinessService* element of the UDDI information model and *operation* and *messagePart* elements that come with WSDL schemas.

Figure 3.6 displays metadata attached to a WSDL message part named *out01* in the WSDL description. The message part metadata is a group of RDF triples stating that the message part has semantic type of *DNA_sequence.*

While the View allows metadata attachments, it is not ontology aware. Even though metadata is attached to various parts of a service description, the ontology that provides the terms used in metadata attachments acts as nothing more than a controlled vocabulary. The relationships between the terms used in metadata attachments are not stored in the View.

As an example consider the ontology term *DNA_sequence* that is used to annotate the output message part *out01* of operation *getSequence* in Figure 3.6. Though this term comes from an ontology in which *DNA_sequence* is defined to be a subclass of *sequence*, such a relationship is not stored or exploited in any way in the View. Therefore our search requests for finding services that produces a *sequence* would not return this service in the result set due the fact that the View does not perform any form of reasoning.

3.3.3.1 Discovery Facilities

The View provides a UDDI-compatible inquiry API that reflects its information model and allows finding Businesses and their services by name and by the tModels that they are categorized by. Additionally it is possible to find services by their metadata attachments. For example, we can find a service that has a metadata attachment of type *qualityRating* that has value of 5*.

The WSDL based inquiry API allows finding operations and message parts by name and more importantly by their metadata. This API allows a View client to pose questions like “Find me a service that has an output message part that has metadata of type *semanticType* and has value of *sequence*”.

To provide more flexibility to the client, the View also supports an RDQL Query interface that allows free form querying of its underlying RDF based data.

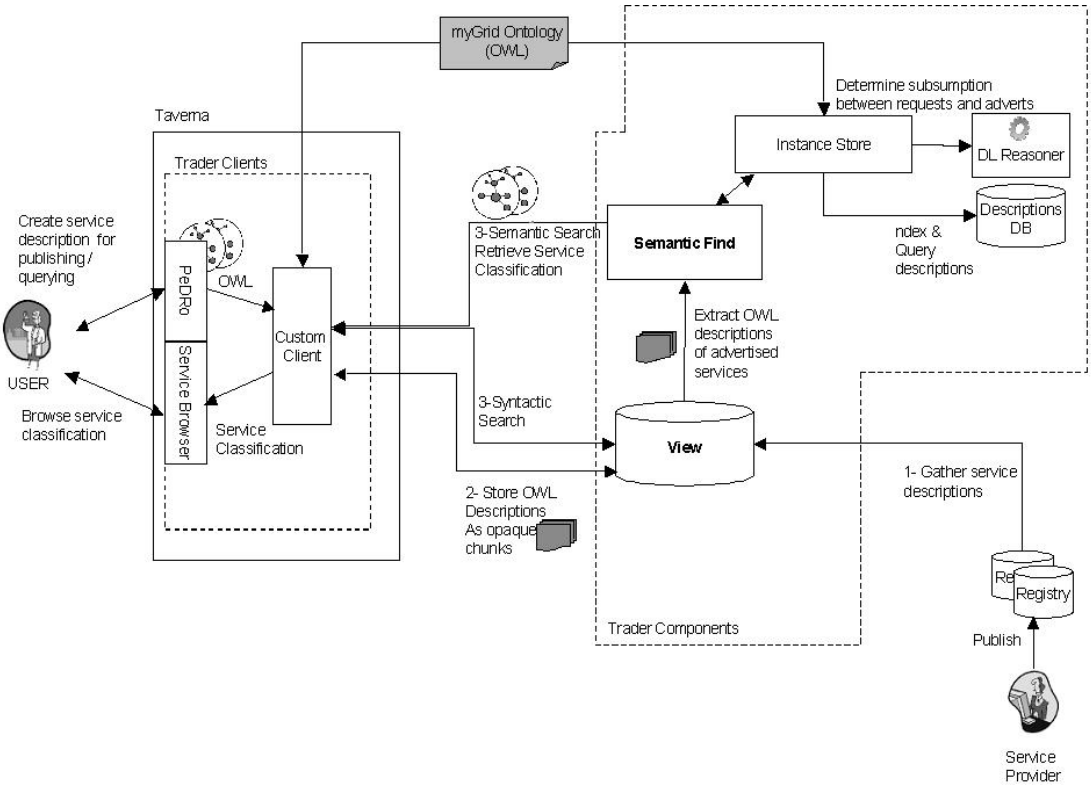


Figure 3.7: Architectural Overview of the Semantic-Rich Approach to Service Discovery

3.4 The Semantic-Rich Approach

The architectural overview of the Semantic-Rich approach is given in Figure 3.7. The components in this architecture can be grouped as Trader Components, Trader Client components, and the OWL based myGrid domain ontology used by both groups.

The **Trader** components, are the View and the Semantic Finds component working cooperatively to provide discovery.

- **The View** has been described previously. In this approach View has been used in a minimal fashion. Neither the WSDL registration nor metadata facilities of the View have been exploited. Only its UDDI compliant interface has been used for the purpose of registration of minimal service information composed of service name and textual description. A more detailed discussion of the extent of the View's usage is given later.

- **The Semantic Find** component makes use of the Instance Store (see Section 2.7.1). Within this architecture the Instance Store is responsible for
  (1) Consuming both the myGrid domain ontology and any additional ontological (OWL based) service descriptions generated during publishing, and performing DL reasoning over descriptions to generate a Service Classification. An example extract of the service classification was given in Figure 2.15 of Chapter 2. The Instance Store has also been responsible maintaining the classification hierarchy as new service descriptions are added.
  (2) Storing and indexing actual service instances with respect to their classification taxonomy in a database
  (3) Accepting service search requests which are OWL descriptions of a desired service, and returning information on matching service instances indexed with that description or its subsumed descriptions.

The **Trader Client** components are: the GUI Pedro annotator; the GUI service browser; and client side custom components that interact with the back-end Trader components based on user requests.

- **Pedro** has been used to generate OWL descriptions of services with respect to the myGrid service schema v.1. The Pedro tool acts as a GUI based service publish/query interface and allows users to generate OWL descriptions to either be submitted to the Instance Store for publishing or be used as a search request for a desired service.
- The **Service Browser** is a GUI based browser that is used to display the hierarchy of service classifications, which is inferred from the service descriptions in the ontology, and each service instance classified with that hierarchy.

In a typical scenario in this approach the following activities take place (see Figure 3.7):

1. UDDI compliant description of services, composed of a BusinessService entity with name, text description and links to a service/workflow interface description (e.g. WSDL doc, Scufl script) are published by service providers to registries. These descriptions are then gathered into the registry View for further annotation and querying.
2. Annotators would generate OWL descriptions for a particular published

service using Pedro. The OWL description is then submitted to the View. However this description is stored as an opaque chunk of information in the View and is not processed by it. So the View only acts as a storage place for these semantic descriptions.

3. Service inquiry requests, again generated by Pedro, are split into syntactic and semantic parts. Syntactic inquiry requests based on keyword based searches on service descriptions are evaluated by the View. The semantic part of the query that contains a domain specific description of the desired service are evaluated by the Semantic Find component. During its initialization, the Semantic Find component retrieves all OWL descriptions from the View and stores them in the Instance Store. To answer a search request, the Instance Store interacts with the DL reasoner to find out where the OWL description of the desired service fits within the classification, then retrieves the corresponding instances from the database.

3.4.1 Remarks on Semantic-Rich Approach

The development and use of the Semantic-Rich approach have led to the following observations:

- The separate evaluation of semantic and non-semantic parts of service search queries by different components has proved to be an extensible discovery framework in which the View component provides domain-independent service registration, personalization and publishing along side a Semantic Find component that is responsible for domain dependent service search.

- By allowing services to be described by highly expressive ontology languages and performing subsumption reasoning over the descriptions during query evaluation; the "Semantic Find" component can be described as a fully fledged semantic discovery component. However the Semantic Find component's use of DLs and DL reasoning (at discovery time) brings about costs in two respects.
  - The first one is the high technical cost, due to computational complexity of DL reasoning [42] which has been experienced in the Semantic-Rich approach.
  - The second one is the cost imposed on the user base in terms of requiring familiarity with description logics constructs. In the Semantic

Rich approach the complexity cost on users has not been experienced since they have been faced with the user-friendly fill-in forms of Pedro that operated over a fixed schema and allowed restricted annotation of these descriptions with named terms from the ontology. When we consider the (current) expectations of users this level of semantics support is satisfactory. The issue then is the fact that DL based formalisms and DL reasoning has been under-exploited since many features of DL specific expressivity such as class union, intersection, complement, universal and existential qualifiers. Such level of semantics support can be provided by less expressive ontology languages and simpler reasoning as we will describe in Chapter 4.

3.5 The View-Only Approach

The architectural overview of the View-Only approach is given in Figure 3.8. In this approach the Semantic Find component has been removed from the architecture and the single **Trader** is the myGrid View service registry. The absence of the Semantic Find component leads to extensive use of the View's functionality for registration of UDDI and WSDL service descriptions and service metadata to support myGrid's Information Model of services. To register information other than a service's name and textual description, the View's WSDL, and metadata publishing/querying APIs are used in addition to the standard UDDI API.

The **Trader Client** Components in the second approach are also the GUI based Pedro tool, a service querying GUI and custom clients. Unlike the first approach, the Pedro tool is not used to generate OWL descriptions of services. Instead, using the myGrid service schema (in Figure 3.3) and the myGrid domain ontology as a controlled vocabulary, Pedro is used to generate service descriptions in the form of its inner record model temporarily. These inner representations of services are later transformed into API calls with respect to the View's UDDI and WSDL based model.

In a typical scenario :

1. Service descriptions were gathered into the View as the first step similar to the Semantic-Rich approach.
2. Capability descriptions of services were published at a second stage, where the View is queried for existing services and the resulting services were

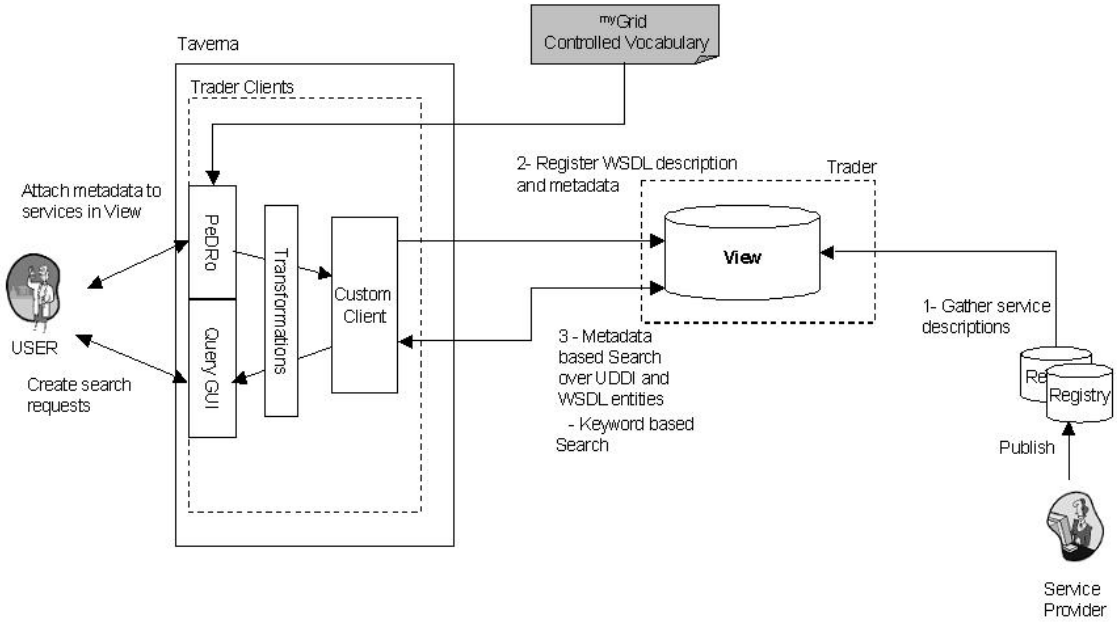


Figure 3.8: Architectural Overview of the View-Only Approach to Service Discovery

further subject to annotation, to provide a domain specific description by using Pedro. The annotated description was then submitted to the View via its WSDL, and metadata APIs. As seen in Figure 3.8, some transformations take place when the descriptions are transferred between Pedro and the View inquiry/publish client modules. This transformation is necessary due to the differences between the information models supported by the user-facing components and the View, which is also depicted in the context diagram in Figure 3.1. The user facing components adopt the myGrid service schema, which models user's view of services and their capabilities, described previously (see Figure 3.3). Whereas the View has a WSDL based model where services are expected to be described by WSDL documents and be registered that way. Entities that need to be discovered that are not necessarily described by a WSDL document, for example workflows and Java objects. More importantly, a service's WSDL description is not necessarily a description of its capability from a users perspective for example Soaplab services. One can argue that the WSDL schema has all the necessary entities that are needed to describe a service its operations and its parameters.

While we acknowledge that it is not impossible to represent myGrid service entities using WSDL schema, we believe that generating artificial WSDL descriptions for a workflow or a local java object is not an ideal way of modelling these entities. Such difficulties were not experienced within the Semantic-Rich approach where both the the user facing components and the Trader adopted myGrid's service schema v.1.

3. Inquiry about services is done via a custom query interface that allows users to search the View to perform name/keyword based and metadata based searches. Metadata inquiries exploited ontology terms attached to services, operations and message parts during annotations. An example would be "Find a service that has input message part metadata of value *sequence*". The role of ontology terms used in metadata attachments in the View is not more than a controlled vocabulary. The View has no notion of relationships between the ontology terms, and does not do any form of reasoning during service search.

3.5.1 Remarks on View-Only Approach

Having described the main characteristics of the View-only approach, the observations made during its development and use can be summarized:

- The strict ties of the View to WSDL for describing service capabilities is limiting. Using WSDL to describe entities such as workflows that have no connection to WSDL is not a desired way of modelling them.

- Within the View-only approach, the Semantic Find component has been abandoned due to its poor exploitation in the first approach. This decision has caused the discovery system to solely rely on the View for providing semantic support during discovery. The benefit that can be gained from the use of an ontology for annotating service descriptions in the View is not more than the benefit that can be gained from using a common vocabulary. The View is not 'ontology aware' and does not provide any form of support for exploiting the rich expressivity of ontology languages during discovery. On the other hand, the View allows arbitrary RDF metadata to be attached to service descriptions, and allows this metadata to be queried. This aspect of the View makes it a critical component for addressing the 3rd party metadata requirements within myGrid.

	Semantic-Rich Approach	**View-Only Approach**
Back-end Model	Hybrid (RDF & OWL)	RDF
Capabilities	myGrid Service Schema V1.	WSDL
Non-functional Aspects	–	–
Binding	–	WSDL
Discovery Mech.	DL Reasoning	View's UDDI + WSDL API
Discovery Purpose	Composition	
Stakeholders	Human Users	
Multi Trader Support	-	-
Temporary Registration	-	-

Table 3.1: Analysis of Previous Efforts With Respect our Survey Categories

3.6 Reflections on the Two Approaches

3.6.1 Profiles of Previous Approaches

The analysis of the previous approaches with respect to the criteria identified in Chapter 2 is given in Table 3.1.

The Semantic-Rich approach is similar to the Traders of top-down Semantic Web Services (SWS) frameworks with its Description Logic based back-end model, and DL reasoning based discovery mechanism (see Table 3.1). We have previously compared the myGrid service schema used within the Trader in the Semantic Rich approach to other SWS modelling frameworks. We have stated that the myGrid model is a subset of these frameworks which solely focuses on the capability layer and omits the process and invocation layers as these are being handled by myGrid's middleware and are not of interest to the user.

The View, hence the View-only approach, overcomes the limitations of UDDI by explicating WSDL in its information model and providing support for metadata. It uses WSDL to describe both service capabilities and service binding details. This representation mechanism can be misleading for entities that are not web services (e.g. workflows) but have been described with the WSDL model since it is the only modelling element in the View.

The myGrid View used in both approaches supports consolidation of contents

of multiple traders into Views, however it does not address coordinated evaluation of service search requests. Similarly temporary registration is not addressed by any of the approaches.

3.6.2 Analyzing Previous Approaches with Respect to myGrid's Requirements

Having described both approaches in detail it is useful to go back to the initial set of requirements outlined in Chapter 1 to be able to judge how each approach meets them. A summary of the requirements is given in Table 3.2.

Following from the table:

1. The View only approach fails to fully meet requirement no 1. regarding a user-oriented capability based model, due to its WSDL based description mechanism within the Trader. The user-oriented view of services as described in the myGrid service schema is abstracted from any particular invocation interface. Therefore using WSDL to describe different types of services is not a solution to this requirement. On the other hand the Semantic-Rich approach supports the myGrid abstract model of service capabilities within the Trader and the user facing annotator and querying components.
2. Requirement 2.a regarding dynamic discovery and composition of shim services is not met by any of the approaches since both aimed at user-oriented discovery for manual composition of experimentally significant services. Requirement 2.b on pro-active discovery and suggestion of successor services in a workflow design context is partially met since the myGrid service schema captures necessary information by providing semantic types for parameters. This information would be enough to answer questions that can be asked by a workflow advisor tool in Taverna. However the mechanism that would pose the question on has not been implemented in any of the approaches.
3. Describing the functionality of a service (3.a) is fully met by the Sematic-Rich approach with its adoption of the myGrid service schema and its use of description logics to represent descriptions and link them to bioinformatics domain knowledge. It is partially met by the View due to its WSDL based model and lack of support for ontologies. Neither of the approaches have explicitly modelled non-functional aspects (3.b) of bioinformatics services.

Req. No.	Description	Semantic-Rich Approach	View-Only Approach
1	User-oriented, workflow centric discovery of different types of services (i.e. operations)	Met	Partially met
2.a	Dynamic discovery of shim services	Not met	Not met
2.b	Dynamic and Proactive discovery of candidate successor services in a workflow context	Partially Met	Partially Met
3.a	Information Model - Service Capabilities	Met	Partially Met
3.b	Information Model - Non-functional Aspects	Not Met	Not Met
3.c	Information Model - Non-restricted 3rd party Assertions	Partially Met	Partially Met
4	Exploiting domain knowledge	Met	Minimally Met
5.a	Discovery by keyword based over name/text description	Met	Met
5.b	Discovery by browsing a service classification hierarchy	Met	Not Met
5.c	Discovery by sending search requests based on the information model	Met	Met
6.a	Deployment - Unified interface for search facilities	Met	Met
6.b	Management of service provider, and third-party descriptions by their owners	Met	Met
6.c	Accessibility from the workflow design context	Met	Met

Table 3.2: The Addressing of Discovery Requirements by Semantic-Rich and View-Only Approaches.

Finally, the View component used in both approaches provides the necessary infrastructure for 3rd party metadata attachments (3.c). However in neither of the approaches have users been provided with mechanisms (e.g. GUI based interfaces) to publish/inquire free-form metadata to service descriptions.

4. By using reasoning during discovery the Semantic-Rich approach meets the requirements, and exploits domain knowledge. The View-Only approach provides minimal semantics support with metadata facilities and the use of ontology as a controlled vocabulary.
5. Regarding desired forms of discovery, all three of them are supported by the Semantic-Rich approach. This provides a classification of services based on their conceptual descriptions, supports keyword based search over the View and answers inquiry requests that partially or fully describe a desired service as an OWL description. Due to its minimal support for semantics, the View-Only approach has no means for providing a classification of services on domain-specific criteria. Keyword based search and inquiry based on a UDDI and WSDL model of services is supported.
6. Both approaches support deployment requirements 6.a and 6.c by providing a unified interface to discovery facilities from within the Taverna workbench environment. The View registry used in both approaches inherently supports owner based management of third-party assertions on services (6.b).

The development of a solution for service discovery in myGrid is certainly an evolutionary process. During the development and use of the two approaches, requirements for service discovery have been progressively articulated.

In summary the two approaches differ in two respects:

1. **The Information Model.** The abstract annotation model of the Semantic-Rich approach is appropriate for describing service capabilities in myGrid. On the other hand the WSDL based information model of the View-Only approach is not suitable for the domains needs.

2. **The expressivity of Semantics, and the nature and timing of reasoning.** The View-Only approach's support for semantics via metadata attachments and a controlled vocabulary is not enough for the domain's needs [57]. The Semantic-Rich approach provides fully-fledged semantics via the use of ontologies and DL reasoning during discovery. However, in practice the users expectations for semantics support is largely on sub-class

relationships among domain concepts. This level of semantics support can be provided by performing simpler forms of reasoning such as RDF(S) reasoning.

Based on these observations we have developed a third discovery facility, called Feta which will be described in the next Chapter.

Chapter 4

Service Discovery in myGrid: Feta Approach

4.1 Introduction

Feta is the most recent service discovery effort in myGrid, which has been developed based on the lessons of two previous approaches. The philosophy behind its design is:

1. To provide semantics support at the medium scale, by only allowing pre-inferred sub-class relationships to be exploited during discovery. This way the domain's needs regarding reasoning and expressivity would be met without imposing complexity in deployment of the system.
2. To adopt the user-oriented model of services captured in the myGrid ontology [57] [95] uniformly, within the Trader and within user-facing components to allow correct modelling of entities in both.

In this chapter we describe the most recent form of myGrid's service schema over which Feta operates. We then give an architectural overview of Feta and its components The operation of the system is analyzed in three main parts. First the generation and annotation of service descriptions with respect to Feta's the information model is presented. Next a detailed view of the mechanism that is used to answer service search requests is given. Finally the interaction of the system with the users during workflow design is given. We conclude with a brief explanation of implementation details and a summary.

4.2 Basis of Feta's Information Model

The desired discovery system within myGrid is one that aids users' decision making process during discovery, not one that provides fully-automated discovery and composition by making the decisions on their behalf. In order to fully support users during service selection, it is essential that the descriptions conform to an information model that reflects their view of entities to be discovered in the system. The aim of the myGrid service schema v.1 , which was adopted by the previous discovery systems was to meet these requirements by

- describing services that are incorporated into workflows as operational steps,
- capturing functional aspects (capabilities) of these services in the descriptions.

Our information model in Feta, which we refer to as myGrid Service Schema v.2 is an evolved version of the myGrid Service Schema v.1. A simplified view of Feta's Information model is depicted in Figure 4.1,

Previously both service and operation entities were allowed to have input/output parameters and can be identified by bioinformatics specific characteristics. The intention was to allow workflows to be described as a service and their operational steps to be described as operations. However, during use of this model it has been observed that annotators become confused with the dualism between service and operation entities when entities other than workflows were to be described. For example, when describing a WSDL based service that has a single operation, the functional unit that needs to be annotated to enable discovery is the operation entity. However users had the impression that it suffices to annotate the encapsulating service entity and did so, which caused these operations to be out of scope during discovery.

Furthermore it is observed that the requirements regarding description of workflow subcomponents, data and control flow is not clear at this stage of the project. Therefore a uniform approach has been taken and the dualism between service and operation elements have been eliminated in Feta by focusing capability attributes around the notion of *operations* only. Analysis of description of the internals of workflows, and their exploitation in discovery has been deferred to future research.

The main elements of the information model is given in the UML conceptual class diagram in Figure 4.1. The main entities are Service, Operation and Parameter that are used collectively to describe capabilities of entities subject to

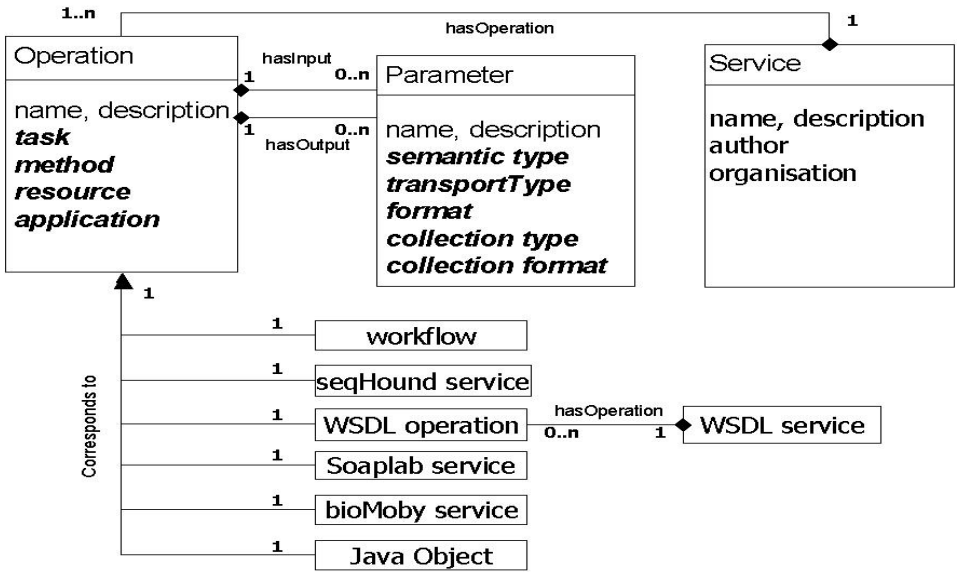


Figure 4.1: A
Conceptual View of the Information Model of Feta. The XML Schema Document corresponding to this model is provided in Appendix A

discovery in MYGrid. The attributes of each element depicted in italic characters are placeholders for ontology terms in service descriptions.

4.3 Feta's Information Model of Services

4.3.1 Modelling Operations not Services

Within our model we distinguish between the core unit of **functionality**, i.e. the **operation** and the unit of *publication*, i.e. the *service*. The *service* entity embodies information on service provider, the author of the description and a textual description of the service.

A service may provide more than one operation. 'Plain' web services are good examples for this. Generally, for services described by WSDL, a single WSDL document contains descriptions of multiple operations even though these operations may be disconnected in terms of functionality. Therefore operations are modelled as a separate entity in our model. The types of operations that can be described are: a WSDL based service's operations, a Soaplab service, a Scufl workflow, a BioMOBY service, a seqHound service, and a local Taverna

compliant Java object.

Stateless	BlastReport doBlast(Sequence, database, program. . .);

Statefull	JobIdentifier createJob();
	void setDatabase(JobIdentifier, Database);
	void setProgram(ObjectIdentifier, Gap);
	. . .
	BlastReport getsomeResults(ObjectIdentifier);

Table 4.1: Two different service interfaces to BLAST, a widely used bioinformatics tool. Differences Between WSDL based(Stateless/Single-Method) and Soaplab based BLAST implementations

The notion of operation within the model corresponds to an operation from the **user's perspective**. The operation entity may not have a one to one correspondence with the operation at the service invocation layer. A good example for this is the Soaplab services which have differences from 'plain' web services. To illustrate this difference the invocation patterns for two different implementations of "Basic Local Alignment Search Tool (BLAST)" are given in Table 4.1. A group of operations need to be invoked consecutively to perform a BLAST task with the Soaplab implementation, whereas only a single operation needs to be invoked to achieve the same task via 'plain' web service implementation of BLAST.

The difference between the invocation patterns of the services is of little interest to the user. These low level service details are handled by the workflow design environment Taverna and its associated workflow enactment engine Freefluo, which provides a common abstraction of these services to the user. It it this abstraction that is being described in our model. Therefore both types of services would be modelled with a single operation entity.

Another important aspect that makes our model different from the previous one (i.e. myGrid Service Schema v.1) is regarding modelling of workflows. Workflows are modelled as single operations in Feta,; the internal steps, data and control flow of workflows are not described. We believe that being able to discover bioinformatics workflows based on their inner working mechanisms (i.e. data and control flow) may be important and probably requires complicated (e.g. fuzzy) matching techniques to be employed during discovery. However requirements for such discovery are still not clear at this stage within the project. Hence our information model excludes description of internal structure of workflows.

4.3.1.1 Attributes of Operations

The operation entity has four attribute types that are placeholders for bioinformatics influenced characteristics of an operation. These are:
(1) the overall *task* being performed by the operation for example pairwise_aligning,
(2) the *method* used to perform that task for example a particular algorithm such as the *Needleman_and_Wunsch_global_sequence_alignment_algorithm*,
(3) the type of *application* used to provide the functionality for example The European Molecular Biology Open Software Suite, EMBOSS,
(4) the *resources* that are used during the execution of the operation for example a Protein Database like SWISSPROT. The information model allows multiple instances of attributes of above types to be associated with a property. For example an operation can be described to perform the tasks of both "alignment" and "sequence retrieval".

4.3.2 Parameters

The parameter entity is used to model both inputs and outputs of operations. A parameter can be described at multiple levels by the use of its attributes. At the highest level of abstraction a parameter can be characterized by a domain concept for describing the *semanticType* (e.g. DNA_sequence). At a lower level bioinformatics specific format of the parameter can be specified via use of the *format* attribute (e.g. AGAVE Format -Architecture for Genomic Annotation, Visualization and Exchange Format-). Even low level *transportType* descriptions that may be of interest to the user can be made (e.g. String). Finally the structure of data that goes in and out of services as parameters can be described via *collectionType* (e.g. Single, Set, Collection) and *collectionFormat* (e.g. Tab_Delimited_Row) attributes.

4.4 System Overview

Feta is not intended to meet all discovery requirements identified in Chapter 1. It has rather been developed as a light-weight domain specific service search facility that focuses on semantic service discovery based on service capabilities.

The architectural overview of Feta is given in Figure 4.2. Following our convention in the previous chapter we group the components in the architecture as

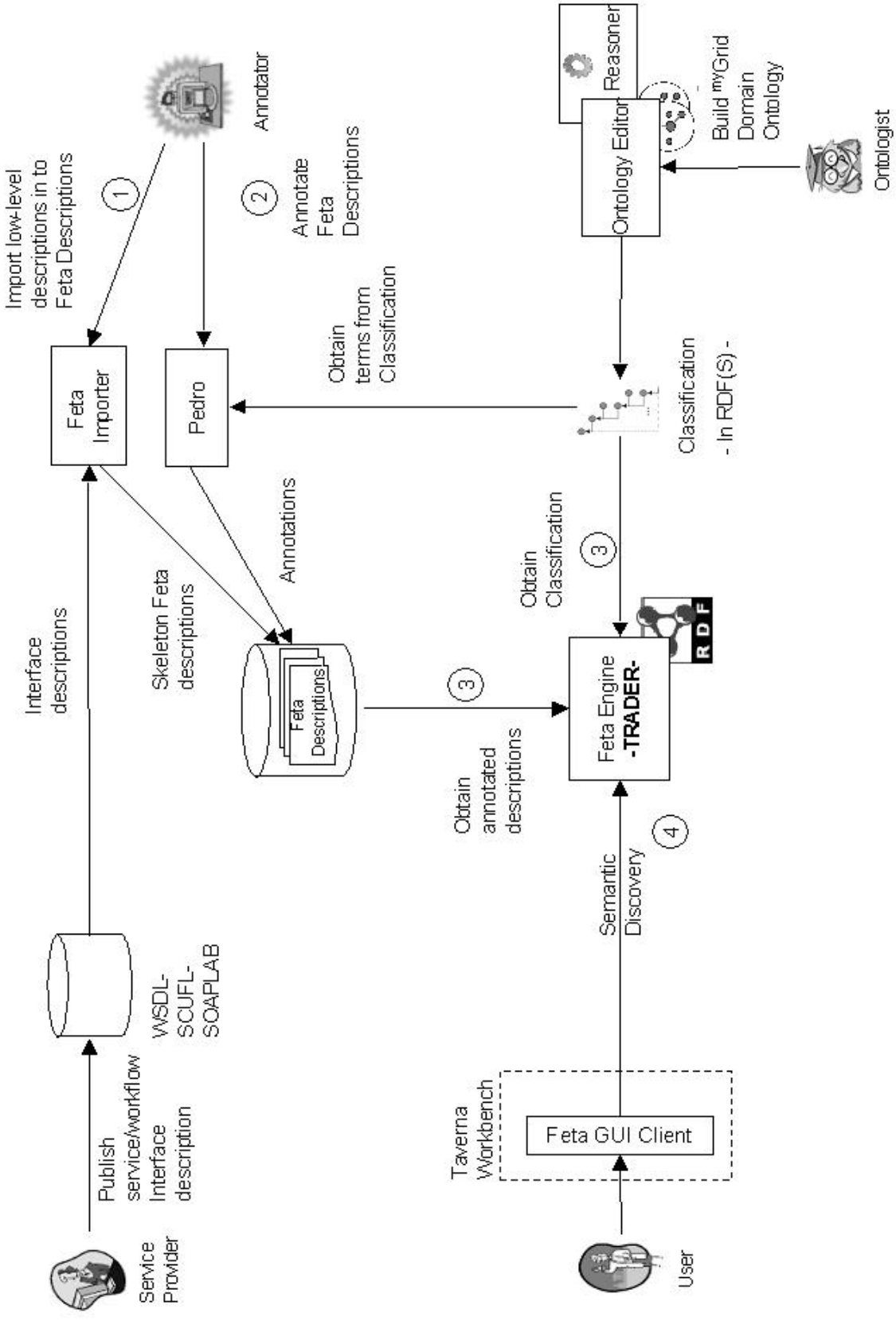


Figure 4.2: Architectural Overview of Feta.

Trader Client (dark-grey shaded) and Trader (light-grey shaded) Components and the myGrid domain classification used by both.

4.4.1 Trader Components

The single Trader component in the architecture is the **Feta Engine**. This component is responsible for loading service descriptions from a designated location, accepting service search requests, evaluating them and returning resulting services. Detailed information on the Feta Engine will be given later.

4.4.2 Trader Client Components

Trader client components, are the Feta Importer tool, Pedro annotator and Feta Taverna GUI plug-in.

- The **Feta Importer** is a group of tools that aid users to generate skeletal service descriptions (in XML format) that conform to the Feta Information Model of services (a.k.a. Feta descriptions). These descriptions are representatives of services that are abstracted away from their invocation interfaces. Even though an abstraction exists it is still possible to transfer some of the information in the invocation specific descriptions into Feta descriptions.
- The role of **Pedro** in Feta's architecture is to enable annotation of skeletal descriptions with terms from the myGrid ontology. Unlike two previous approaches, in Feta the Pedro tool has been used in its default configuration to populate **XML** descriptions with either user-provided values or terms from the myGrid ontology.
- The **Feta Taverna Plug-in** is a GUI tool used by workflow designers from within Taverna for making search requests for services, viewing the results and incorporating them into workflows.

4.4.3 myGrid Domain Classification

The final component in Feta's architecture is the myGrid domain ontology, which is in the form of a classification in RDF Schema. RDF Schema, which has served as a basis for development of many ontology languages, can be used to represent domain knowledge in a restricted but simplified manner. The expressivity provided by ontology languages like OWL have turned out to be unnecessary for

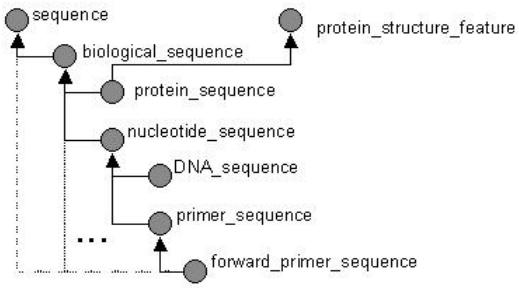


Figure 4.3: An Extract of Classifications in the Simplified myGrid Domain Ontology. The inferred sub-class relationships evaluated during service search are depicted as dashed lines. Appendix B contains an extract from the lite version of the classification.

describing bioinformatics resources within myGrid. Therefore, RDF(S) is an ideal language for providing medium level semantics support.

In our approach we use neither the Description Logic versions (i.e. DAML+OIL and OWL versions) of the myGrid ontology nor the DL Reasoners during discovery. Instead the myGrid ontology in OWL is reasoned over by a DL reasoner and the resulting classification hierarchy is stored in RDF(S) retaining only the classes, their properties and class and property hierarchies. Two versions of RDF(S) based snapshots of the ontology have been used within Feta. The sizes of these are given in Table 4.2. The first one of these, which is the first-cut RDF(S) export of the DAML+OIL ontology, has larger number of classes and sub-class relationships. While this classification is comprehensive most of the classes in it are not expected to be used during annotation. Therefore a cut-down (lite) version of the classification has also been generated by taking a subset from the larger classification. This subset contains the classes that are expected to be more used during annotation.

The RDF(S) based domain ontology does not contain a classification for services generated from their ontological descriptions. An example of this classification was given in the previous Chapter in Figure 2.15. The RDF(S) classification used in Feta covers:

- The "service sub-ontology". This introduces classes such as *service*, *operation* and *parameter*, and properties such as *hasInput*, *hasOutput* and *hasOperation*, that can be used to describe a service.
- The 'molecular biology sub-ontology'. This contains classes such as *DNA_sequence*,

Sub-Ontology	Full Classification			Lite Classification		
	# of Classes	*# of Properties*	*# of Sub-Class Rel.*	*# of Classes*	*# of Properties*	*# of Sub-Class Rel.*
Service	9	28	0	9	28	0
Molecular Biology+ Bio-informatics+ Informatics.	549	39	708	129	0	127

Table 4.2: Sizes of Different Versions of myGrid Domain Classification

BLAST_Report and the sub-class relationships between these classes. These classes are commonly used to characterize inputs and outputs of services.

- The 'bioinformatics sub-ontology', which contains classes such as *aligning*, *retrieving*, *SWISS-PROT*, *EMBOSS* and their subclass relationships. These classes are used to characterize methods, tasks and the accessed resources of a service in myGrid.

4.4.4 System Operation

The interaction of components within the architecture given in Figure 4.2 is as follows:

(1) The invocation interface descriptions of services (e.g. WSDL descriptions, Scufl workflow scripts), which are published by *service providers*, are consumed by the Feta Importer tool to generate skeletal Feta descriptions;

(2) These descriptions are further annotated with the myGrid domain ontology terms by *annotators* using Pedro;

(3) Once the skeletal descriptions are populated and annotated they become ready to be queried over. Annotated descriptions are loaded in to the Feta Engine together with the myGrid domain classification. The Feta engine is then responsible for answering *users'* search requests by taking the sub-class relationships between annotation terms in to account. The users are provided with a GUI tool that allows them to choose from a list of pre-defined query templates, to fill them in with desired values and to submit them to the engine.

Now we will elaborate on each of the steps that take place during the operation of Feta.

4.5 Capability Publishing

As mentioned earlier our goal in Feta is to enable discovery of services based on their capabilities. Given that a service's capability is not reflected in its invocation interface description published by the service providers, it becomes essential for a secondary publishing stage to take place. During this stage a capability description for a service is generated in addition to its interface description. Service re-publishing can be performed by service providers or annotators who are specialized in generating domain specific descriptions of services.

4.5.1 Generation of Feta Descriptions

To ease the process of generating service descriptions and to lower the activation energy needed to start up a discovery system, the users are provided with specialized importer tools that generate annotation-free skeletal Feta descriptions, using different types of low-level invocation interface descriptions. The importer tool extracts information such as service name, textual description, parameter names, and uses them to generate annotation free, skeletal Feta descriptions.

The reason for choosing XML as the description representation language is due to its widespread use and strong tool support. Furthermore XML does not introduce any restrictions on semantic annotations of services. The domain ontology to be used for describing and discovering services is represented in a simplified ontology language RDF(S) where there are only named classes and hierarchical relationships between those classes. Therefore the annotation process is only expected to consume these named classes and attach them to different parts of skeletal XML descriptions.

In its current state the Feta Importer supports automated skeleton description generation for WSDL based services and Soaplab analysis services and Scufl workflows.

4.5.1.1 Feta Descriptions for Plain Web Services

In the case of "plain" web services where each operation in a WSDL document corresponds to an operational task in the workflow, a one to one mapping between WSDL operations and Feta operations is performed during import. The upper part of Figure 4.4 displays a simplified view of the WSDL description of the DDBJ implementation of the BLAST service from our previous example. The

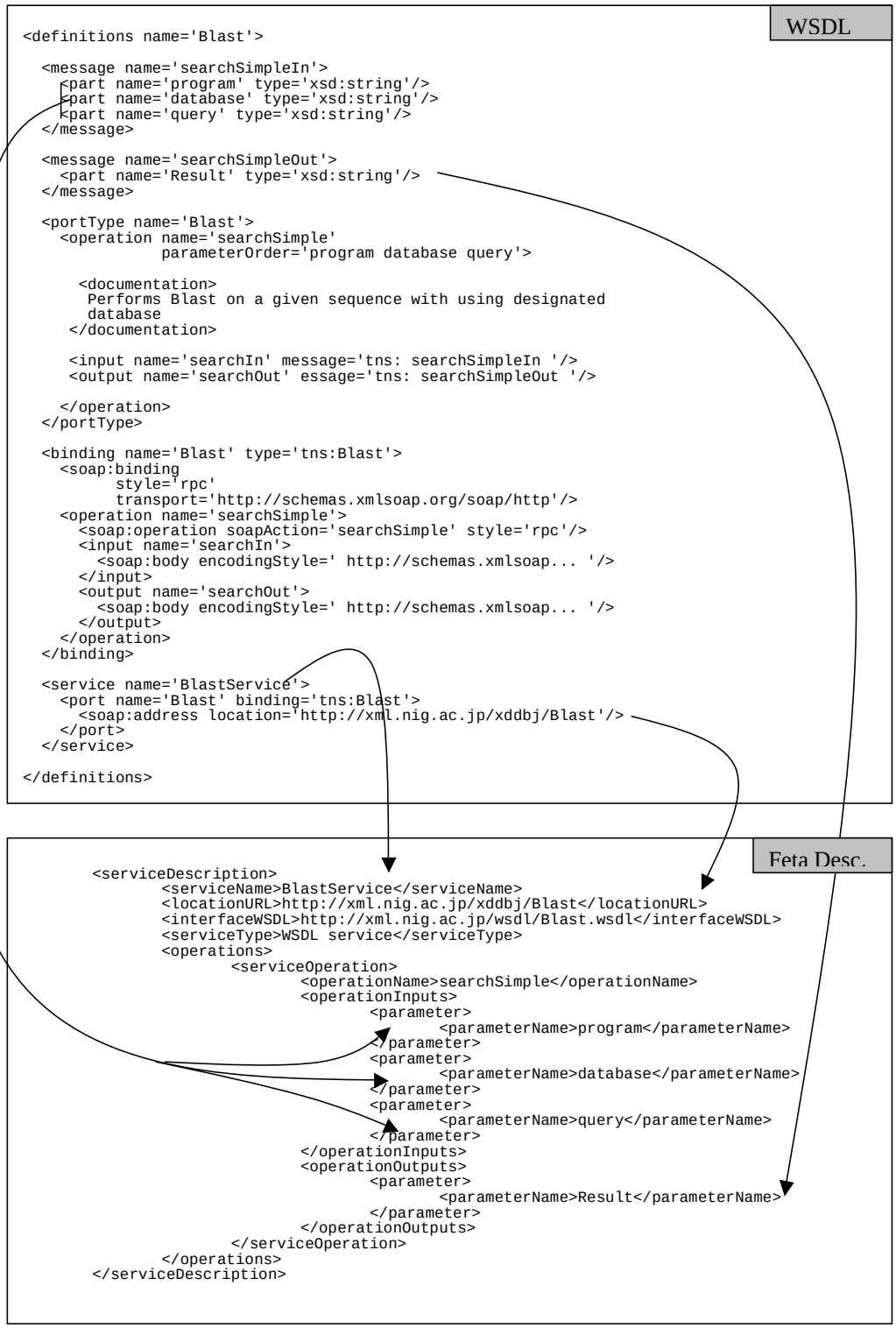


Figure 4.4: WSDL Description of BLAST Service and its Corresponding Skeleton XML Description.

lower part of the figure displays the skeleton XML description generated by using this WSDL file. The complete list of mappings performed during import of WSDL descriptions is as follows:

- A WSDL service element is mapped to a service in the Feta description.
- The SOAP address of the port from which the service is delivered is mapped to a locationURL element.
- Each operation that has been defined in the portType that the service's port adheres to is mapped to an operation in Feta (see portType element with name BlastPortType).
- Each message part that makes up the input output messages of the operations is mapped to a parameter entity.

In addition to these mappings the serviceType element of the Feta description is populated indicate that the service is a plain web service.

4.5.1.2 Feta Descriptions for Soaplab Services

For Soaplab services, whose WSDL descriptions reflect their job control interface rather than an "atomic operational step", the WSDL descriptions are not used for generating skeletons.

Figure 4.5 displays a simplified view of the WSDL description of the Soaplab BLAST service which contains descriptions for multiple operations and messages for operations named "createJob", "run", "getSomeResults". If this WSDL description were to be used, the resulting Feta description would be confusing for end users since it involves operations and intermediary parameters that are related to the stateful invocation pattern of a Soaplab service.

Therefore, to generate skeletal descriptions for these services a specialized catalog service, named Soaplab Analysis Factory Service is used. The catalog service is itself a Soaplab service that provides information about other services hosted on a particular site. The skeleton description for BLAST generated by obtaining information from the catalog service is shown in Figure 4.6. The description is composed of a **single service** entity and a **single operation** entity beneath it for each Soaplab services. The Soaplab catalog service also provides information on the overall number of input and output parameters of the service such as names and default values (see Figure 4.6).

```
. . .

<wsdl:message name="createJobRequest">
  <wsdl:part name="in0" type="apachesoap:Map"/>
</wsdl:message>

<wsdl:message name="createJobResponse">
  <wsdl:part
        name="createJobReturn" type="xsd:string"/>
</wsdl:message>

  <wsdl:message name="runRequest">
    <wsdl:part name="in0" type="xsd:string"/>
    <wsdl:part name="in1" type=" xsd:string "/>
  </wsdl:message>

  <wsdl:message name="runResponse">
    <wsdl:part name="runReturn" type=" xsd:string "/>
  </wsdl:message>

    . . .

  <wsdl:portType name="AnalysisWSAppLabImpl">
    <wsdl:operation name="createJob" parameterOrder="in0">
      <wsdl:input
                 message="impl:createJobRequest"
                 name="createJobRequest"/>
      <wsdl:output
                 message="impl:createJobResponse"
                 name="createJobResponse"/>
    </wsdl:operation>
    <wsdl:operation name="run" parameterOrder="in0 in1">
      <wsdl:input
                 message="impl:runRequest"
                 name="runRequest"/>
      <wsdl:output
                 message="impl:runResponse"
                 name="runResponse"/>
    </wsdl:operation>

    . . .

  </wsdl:portType>

. . .
```

Figure 4.5: WSDL Description of Soaplab BLAST Service.

```
<serviceDescriptions>
  <serviceDescription>
    <serviceType>Soaplab service</serviceType>
    <serviceName>seq_analysis::blast2</serviceName>
    <locationURL>
http://phoebus.cs.man.ac.uk:8081/axis/services/seq_analysis::blast2
     </locationURL>
    <interfaceLoc>
http://phoebus.cs.man.ac.uk:8081/axis/services/seq_analysis::blast2
?wsdl
</interfaceLoc>
    <operations>
      <serviceOperation>
        <operationName>seq_analysis::blast2</operationName>
        <operationInputs>
          <parameter>
            <parameterName>sequence1_direct_data</parameterName>
            <transportDataType>String</transportDataType>
          </parameter>
          <parameter>
            <parameterName>program</parameterName>
            <transportDataType>String</transportDataType>
          </parameter>
          <parameter>
            <parameterName>database</parameterName>
            <transportDataType>String</transportDataType>
          </parameter>
          <parameter>
            <parameterName>querystrands</parameterName>
            <transportDataType>String</transportDataType>
          </parameter>
          . . . .
          . . . .
        </operationInputs>
        <operationOutputs>
          <parameter>
            <parameterName>report</parameterName>
            <transportDataType>String</transportDataType>
          </parameter>
          <parameter>
            <parameterName>detailed_status</parameterName>
            <transportDataType>String</transportDataType>
          </parameter>
          <parameter>
            <parameterName>output</parameterName>
            <transportDataType>String</transportDataType>
          </parameter>
          <parameter>
            <parameterName>seqannot</parameterName>
            <transportDataType>String</transportDataType>
          </parameter>
        </operationOutputs>
      </serviceOperation>
    </operations>
  </serviceDescription>
```

Figure 4.6: Skeleton XML Description Generated for Soaplab BLAST Service

4.5.1.3 Feta Descriptions for Scufl Workflows

Workflows are modelled as single operations. For a particular workflow the importer tool processes the Scufl script and generates a single service entity embodying a single operation entity. The overall inputs and overall outputs of the workflow are mapped to parameters that are attached to the single operation representing the workflow.

While Feta currently provides importer facilities for three types of services the system does not restrict its discovery facility to these three types of services. Descriptions for seqHound services, bioMoby services and local java objects can also be subject to discovery as long as their Feta descriptions exist.

4.5.2 Annotation of Feta Descriptions

Once the skeleton descriptions are generated they need to be annotated to bring in semantics to the description. The skeleton descriptions are annotated via use of Pedro. The schema that drives the operation of Pedro in our approach is Feta's information model expressed in XML Schema. the product of the annotation process is a Feta description (see XML fragment in Figure 4.8) with ontology terms (named concepts) embedded in certain parts as field values.

4.5.3 Publishing of Annotated Descriptions

Currently within Feta's architecture, there is no push-based publishing of annotated service descriptions to the Feta Engine. The publishing is a decoupled process where annotated Feta descriptions (in XML) are made available to a web accessible location or a local file store, from where they are picked up and loaded in to the Feta Search Engine.

4.6 Feta Search Engine

Once the descriptions are generated and annotated they become ready to be used within the Trader to enable discovery. Feta uses RDF(S) and RDQL to

1. To merge the service descriptions with the domain ontology and
2. To enable searching over these descriptions.

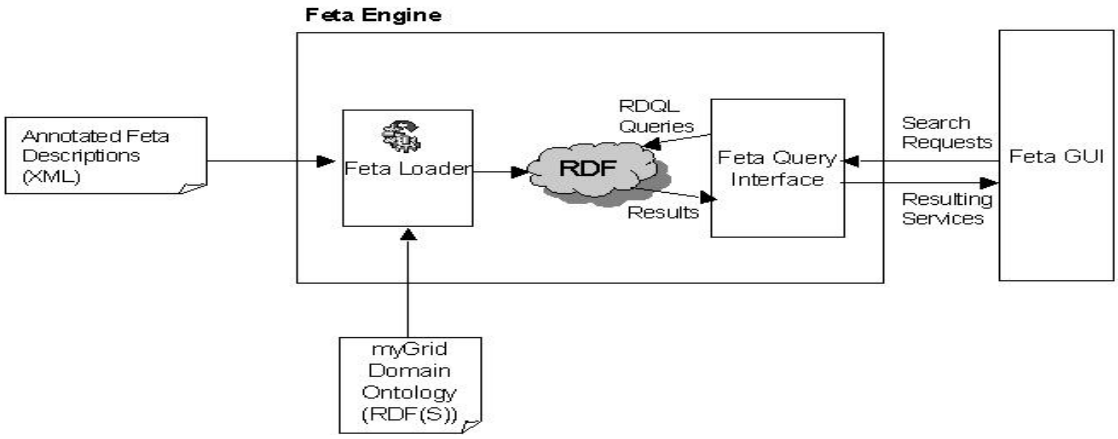


Figure 4.7: A Closer View of The Discovery Engine.

4.6.1 Merging Descriptions and Domain Ontology

Annotated Feta descriptions are merged with the domain ontology by means of a conversion process. Feta descriptions (in XML format) are converted to RDF. The reasons for choosing RDF as the internal representation mechanism are :

- RDF, together with its schema RDF(S), allows a unified representation for both the service descriptions and the ontology that is used to annotate the descriptions.
- RDF and its associated query languages such as RDQL comes with relatively mature tool support [19] [4] that allows RDF(S) entailment capabilities to be automatically exploited while querying RDF with RDQL.

Figure 4.7 provides a view of the inner working mechanisms of the Feta Engine. The domain ontology and descriptions are both fed into the loader component which builds the RDF representation of service descriptions that conform to an RDF Schema in alignment with the schema of XML descriptions. This RDF Schema is in fact the service sub-ontology of the myGrid ontology.

4.6.2 Converting XML Descriptions to RDF

The conversion process from XML to RDF changes the representations of descriptions retaining the Information Model. Five types of mapping rules that are used to transform different entities in our XML Schema. The rules can be explained by following the example given in Figure 4.8:

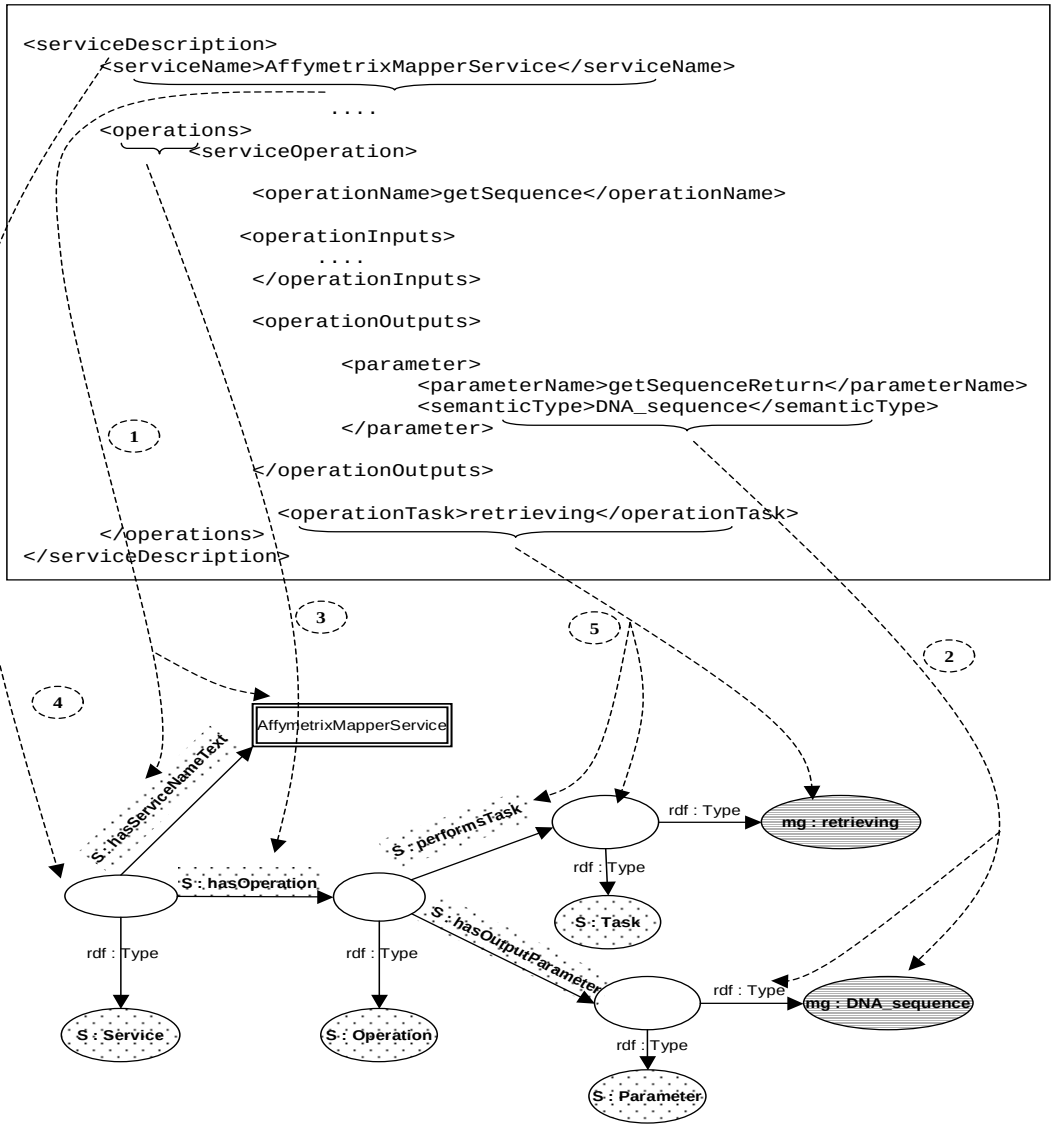


Figure 4.8: Sample XML Description and its Corresponding RDF Representation. The resources (nodes) and properties (arcs) in the figure with dotted shade come from the service sub-ontology. The nodes shaded grey come from the biology, informatics and bioinformatics sub-ontologies.

1. This transformation is used for converting entities that are not placeholders for ontology terms. The body text of an element can be mapped to an RDF plain literal. The literal is connected to the root resource (i.e. resource generated corresponding to the parent XML node of the current node) via properties of certain types for certain element names. The body text of the "serviceName" element is converted to a literal and it has been attached to the root resource via property "hasServiceNameText".

2. The body text of an element may be mapped to an RDF resource. The resource is connected to the root resource via properties of certain types for certain element names. The body text of "semanticType" element is mapped to a resource -a resource corresponding to an ontology term in our example - and it has been attached to the root resource via property "rdf:Type".

3. An XML element without body text containing sub-elements may be mapped to an RDF property. The property is used to connect the resource corresponding to the root XML element to other resources that may result from conversion of child nodes the current element. The "operations" element is mapped to property named "hasOperation".

4. An XML element without body text containing sub-elements may be mapped to a special type of RDF resource (a blank node of a certain specified class/type). The "ServiceDescription" element has been converted to a blank node resource of type "Service" of the service ontology.

5. An XML element with body text can be mapped to two RDF resources, one identified by the body text and the other being a blank node of a certain type decided by the element name. The resource corresponding to body text is attached to the other resource as a type descriptor, and the resource corresponding to the element name is attached to root resource via a property of certain type decided by element name. The body text of "operationTask" element is converted to a resource corresponding to an ontological term "retrieving". A blank node of type "Task" has been generated and linked to the root resource via property named "performsTask".

```
SELECT
      ?descLoc, ?servName, ?opName
WHERE
      (?s mg:hasServiceDescriptionLocation ?descLoc)
      (?s mg:hasServiceNameText ?servName)
      (?s mg:hasOperation ?op)
      (?op mg:hasOperationNameText ?opName)
      (?op mg:outputParameter ?par)
      (?par rdf:type mg:DNA_Sequence)
USING
        mg for <http://www.mygrid.org.uk/ontology#>
```

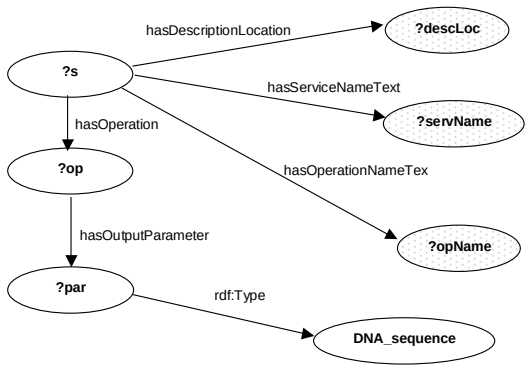


Table 4.3: A Sample RDQL Query and the Graph Pattern it Specifies

4.6.3 Querying Feta Descriptions

Once the descriptions are converted to RDF and merged with the domain classification they become ready to be queried over. Feta makes use of RDQL for providing service search functionality.

An example RDQL query corresponding to a search request for services that have an output with semantic type *DNA_sequence* can be seen in Table 4.3. The WHERE clause within the query specifies a graph pattern for the RDF representation of the desired service.

The particular RDF framework used within Feta, namely Jena [19], inherently takes the RDF(S) entailments into account during evaluation of RDQL queries. Hence the example query given in Table 4.3 would not only return services whose output parameters are of type *DNA_sequence* but also would return those whose parameters are of a type which is a defined to be a sub-class of DNA_sequence in the myGrid domain classification.

4.6.4 Feta Canned Queries (Feta API)

In order to be able to build such RDQL queries, the users should be fully aware of the schema that has been used to build the RDF graph so that they can unambiguously interpret the results. While the ability to generate free-form RDQL queries provides flexibility to users of a system it may be unappropriate since not all system users may (or want to) be aware of the data model (i.e. schema) of the descriptions.

Hence Feta enables the users to make searches along a number axes with the use of pre-canned queries. By using of RDF(S) entailment Feta can answer the following service search requests:

- Find an operation that accepts input of semantic type "X" or something more general.
- Find an operation that produces output of semantic type "Y" or something more specific.
- Find an operation that performs task (or uses method or uses resource or is part of Application) "X" or something more specific.
- Find an operation that is of type "WSDL based Web Service Operation", "Soaplab Service", "Scufl Workflow" etc.
- Find an operation whose name/description contains a certain phrase.

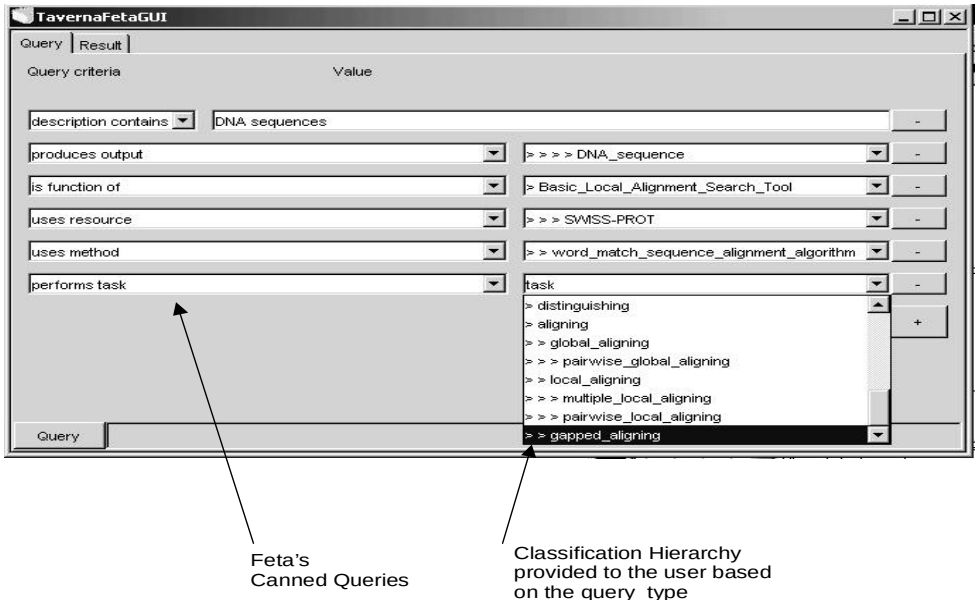


Figure 4.9: GUI Panel for Building Search Requests.

The above set of canned-queries has been identified by a domain expert to be the ones that would be most frequently used for discovery.

4.7 Taverna Feta Plug-In

It is essential to provide access to service search facilities from within the same environment as workflows are built. The overall aim of discovery in myGrid is to support the bioinformatics workflow (i.e. experiment) design process. Therefore Feta's service search capabilities are exposed to the user via a graphical user interface plug-in integrated to the Taverna workflow workbench.

When the user begins designing a new workflow, they launch the service discovery plug-in, which provides query building and results displaying functionality.

4.7.1 Query Building

Figure 4.9 shows the query panel of the Feta plug-in. The query interface enables the user to build a composite service search request that is made up of the canned queries that the Feta engine supports. The users can create as many canned queries as they want. Building a query is achieved by first selecting the canned query type and then providing the value that the desired service should possess

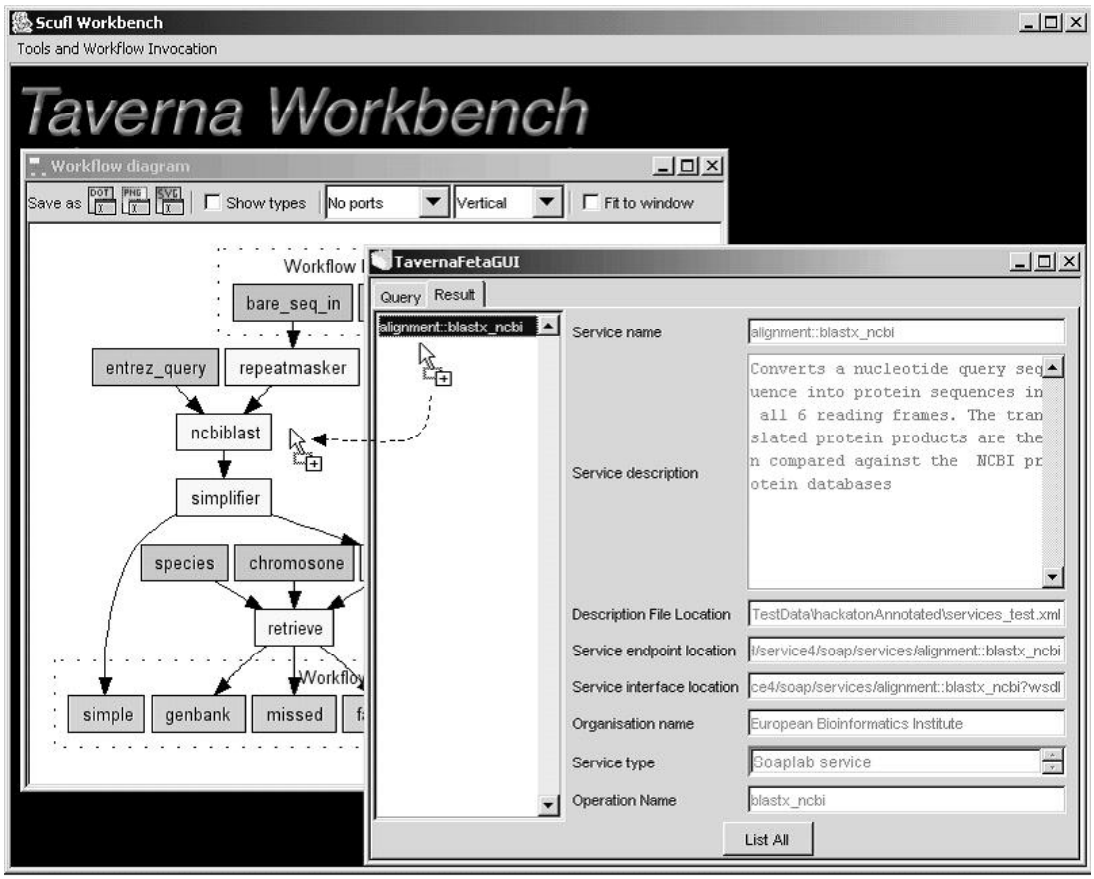


Figure 4.10: GUI Panel for Displaying Search Results

in its description. Depending on whether the canned query contains placeholders for ontology terms or not, the user is either allowed to enter free text criteria (i.e. search requests on service name and description) or is provided with a controlled vocabulary in the form of a hierarchy of ontology terms that could have been used during the annotation of the certain field. The service search request is a conjunctive composite of all individual canned query requests created by the user.

4.7.2 Results Displaying and Results Integration to Workflow

The results of the search are displayed in the Feta Results Panel as seen in the screenshot in Figure 4.10. The results are the list of service entities that may be used as an operational step within a workflow. In addition to the list of resulting services the panel also displays additional information for each entity:

their name, textual descriptions, the location of their Feta description, service end point location and service type, supplying organization's name, service end point location and service's low level interface description document's location.

To incorporate the resulting services into the workflow, services can be dragged and dropped on to the workflow editing panel of Taverna. The workflow editor is capable of recognizing and processing low-level interface descriptions of different types of services, therefore the information that is transferred over to the workflow editor during drag and drop (DnD) is the URL of the interface description document for the service that is the WSDL file URL for WSDL based services, a Scufl file URL for workflows. After the DnD operation is complete the operation is recognized as an operational step within the worklfow diagram panel as seen in Figure 4.10.

4.8 System Implementation

Feta has been implemented in Java using Java SDK 1.4.2-04 Build 05. JFC/Swing GUI classes have been used for development of the workbench GUI plug-in. The Jena RDF toolkit V.2.1 has been used for storing and querying RDF data. Jena provides two options for storing RDF models, namely "in memory" and "persistent over RDBMS" options. Currently Feta operates on an "in-memory" model for better performance. For communicating with Soaplab catalog service classes within Axis V.1.1 package have been used.

In its current implementation Feta is provided as a plug-in application within Taverna workbench. Both components operate on the same Java Virtual Machine hence eliminating any specialized mechanism for interprocess communication.

The source code for Feta is publicly available at http://cvs.mygrid.org.uk/feta/

Future work regarding the Feta implementation is given in Chapter 5.

4.9 Evaluation

In this thesis we are not providing a performance analysis on the query evaluation of Feta Engine. This is mainly because such an effort would be an evaluation of the particular underlying back-end technology that we currently use, namely Jena, and its RDF(S) inferencing mechanism for which a performance analysis has already been provided [18].

To date Jena back-end has proved acceptable performance with respect to our needs. However, in Feta we are aiming at abstracting away from the underlying technology so that RDF implementation frameworks other than Jena [4] can be used as a back-end to Feta. Therefore we avoid providing performance evaluation on a back-end technology, which may be swapped with alternative technologies.

Feta's GUI plug-in to Taverna workbench has been qualitatively evaluated by interviewing with its potential users. The users referred to here are scientists running *in silico* bioinformatics experiments (i.e. myGrid workflows) to characterize a deleted section in a complex region of Chromosome 7 in the human genetic map [82]. This deletion causes the clinical condition known as Williams-Beuren Syndrome. Characterization of a deleted region to produce a complete genetic map requires repeated application of a range of standard bioinformatics techniques to process genome information.

We have obtained the following feedback on Feta's GUI search interface:

- Users build a search request by incrementally creating a group of query criteria. While this brings simplicity to the GUI interface, the users have expressed their preference to build a search request in a form filling fashion where all possible criteria types are displayed on the form and only the criteria values need to be supplied by the user.

- An implicit conjunction operator is applied to the set of query criteria that has been created. The users have stated that they would also like to be able to combine query criteria with other logical operators such as disjunction.

- The information model of Feta does not address non-functional properties of services. However, as we will discuss in the next chapter, more research and development in this area is planned to be done to enable description and exploitation of such service aspects during discovery. While not detailing any exact requirements for non-functional aspects that are of interest to them, the users have stated that they would like to be able to create query criteria on such aspects of a service (e.g. reliability rating greater than 0.6) using the query panel.

- The services resulting from search requests are not ranked. This is because the Feta Engine performs exact matching of services and there is no quantitative aspects of services that could be used for ranking. The

users have stated that they would like to be able to specify ranking criteria to be applied search results based on quantitative service characteristics. For example users would like result ranked descending with respect to a non-functional aspect such as reliability rating.

- The results panel displays information on service name, type, textual description, service and point, and service low-level description (Scufl script or WSDL doc) location. The users also expressed need to display detailed service information such as service annotations, a service's input/output parameters and their associated annotations.

4.10 Chapter Summary

In this chapter we have described the user-oriented semantic service discovery component, Feta, that we have developed within the myGrid project.

Similar to previous approaches in myGrid, the main objective behind Feta's conception is to support users during the design of bioinformatics workflows by providing discovery of workflow building blocks. As a consequence of this, Feta operates over myGrid's service schema (see Table 4.4), that reflects a user's point of view of services rather than an invocation specific low-level view.

Based on previous experience on poor exploitation of domain knowledge expressed in Description Logics during discovery, Feta has experimented with the use of a simplified form of the myGrid domain ontology for discovery. The simplified form is a classification hierarchy (in RDF(S)), that is generated from the ontology (in OWL). Use of a classification instead of a Description Logics based ontology eliminates the need to use DL reasoning for discovery. This aspect of Feta is similar to the bottom-up approaches to Semantic Web Services, where simple sub-class or is-a hierarchy crawling was performed during discovery [59] [91].

To help with the generation of domain-specific descriptions of services Feta provides utility importer tools that mine usable information from low-level descriptions and generate skeletal Feta descriptions.

The incorporation of classification terms into the Feta descriptions is done via an annotation process, which is achieved by use of an external tool Pedro by service annotators.

Annotated descriptions are converted to RDF format for storage and querying within the Trader in Feta's architecture. RDF based descriptions are queried via

	Feta Approach
Back-end Model	RDF
Capabilities	myGrid Service Schema V2.
Non-functional Aspects	–
Binding	–
Discovery Mech.	Feta API + RDF(S) Reasoning
Discovery Purpose	Composition
Stakeholders	Human Users
Multi Trader Support	-
Temporary Registration	-

Table 4.4: Analysis of Feta With Respect to Our Survey Categories

use of RDQL within the Feta trader. While the Trader uses RDQL for its internal evaluation of service search requests, it exposes an interface that reflects a set of pre-defined search requests (canned queries) commonly used to search for services in myGrid. Furthermore the search functionality of Feta is semantically enhanced via incorporation of RDF(S) reasoning into query evaluation.

Following its user-orientation objective, the search facilities that Feta provides are made accessible through the use of a GUI plug-in to the Taverna workbench.

Feta is built as a search facility rather than a registry of services therefore it does not support federations or temporary soft-state registration.

Further discussion on Feta, regarding its relation to myGrid service discovery requirements, its intended use and its future extensions are given in the following chapter.

Chapter 5

Conclusions and Future Work

Throughout this thesis we have stated that the motivations that have led to development of Feta were: (1) to support a user-oriented model of services and (2) to provide a medium-scale semantics for discovery. Having described all three approaches to service discovery in myGrid we can position each in Figure 5.2:

- *Expressivity of descriptions.* In Feta service descriptions are represented in RDF(S) which provides less expressivity than a description logic but more expressivity than a controlled vocabulary. As depicted graphically in Figure 5.1: With a DL ontology, as used in the Semantic-Rich approach, we have the expressive power to generate a common vocabulary for concepts, a set of relations among concepts, and more importantly the flexibility to generate new concepts from existing ones using relationships. Such flexibility is especially useful for building a service classification that defines subsumption relationships between conceptual descriptions of services. With an RDF(S) classification we only have the expressive power to use hierarchically organized named concepts, which are related to certain parts of a service description in Feta. With a common vocabulary (CV), terms of which correspond to named concepts of a classification, we have the minimal expressive power to provide descriptive information about a service. In case of the use of a CV, as done in the View-Only approach, the semantic description for a service is just a group of terms attached to different parts of a description.
- *The timing and type of reasoning.* Unlike the Semantic-Rich approach and many other Semantic Web Service discovery systems surveyed in Chapter 2, Feta employs DL reasoning only at the time of ontology development,

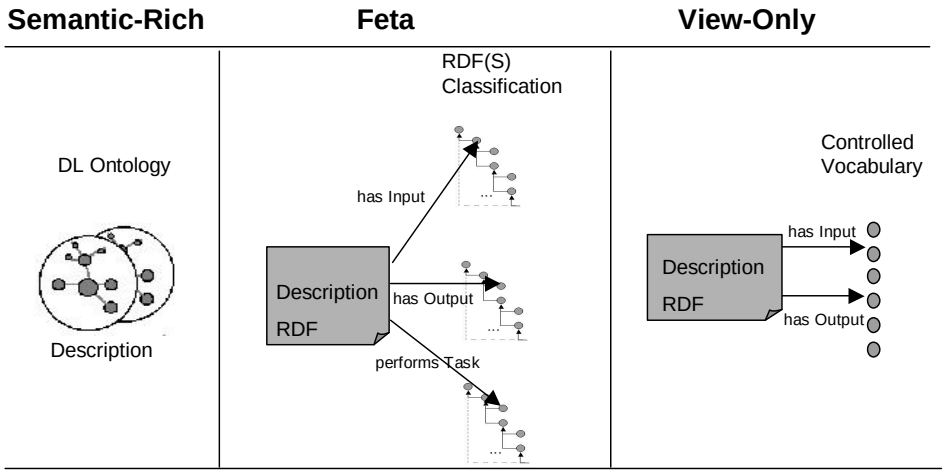


Figure 5.1: A Summary of Three Service Discovery Approaches in myGrid.

and RDF(S) reasoning at the time of discovery (see Figure 5.2).

- *Complexity of Deployment.* There is a benefit in giving up on DLs and DL reasoning at the time of discovery, which is low complexity in deployment which can be observed in Feta and View-only approaches. When DLs and reasoning are used for discovery the scalability of the system reduces due to computational complexities of DL reasoners. Use of DLs and Reasoning has caused the Semantic-Rich approach to have high complexity in deployment, where users have been faced with the execution of a DL reasoner at discovery time. The scalability issues can be coped with if the DL expressivity is essential in a system. However in myGrid the capability to describe and discover services through flexible conceptual descriptions in which all DL concept constructors are used is not essential. DLs bring about complexity in two respects, these are:
  - The performance complexity of reasoning done during answering search requests. As we have surveyed in Chapter 2, hybrid Traders that employ DL reasoning together with certain performance enhancing Information Retrieval techniques are beginning to be deployed in real world settings. While not using DLs for the time being by basing our back-end model on RDF we leave the door open for extensions for DL reasoning since most RDF frameworks including Jena have support for it.
  - The complexity involved in formulating search requests using rich DL

concept constructors. Even when the performance aspects of DL reasoners are overcomed by certain auxiliary techniques and restrictions it is still not feasible to expect bioinformatics workflow designers to build logical formulas (even with use of UI based guidance tools [52]). There is recent promising research on frameworks that provide natural language based representation of knowledge representation formalisms [88]. Such frameworks can be used in the myGrid semantic discovery setting to ease the process of generating DL based service descriptions and discovery requests as and when a requirement for DL expressivity becomes essential.

- *Reflecting the user-oriented view of different types of entities that can be operational steps in a workflow.* The only approach that fails to achieve this goal is the View-only approach, due to its strict ties to WSDL. Feta has followed the path set by the Semantic-Rich approach and adopted a user-oriented view of services in its information model. This high-level model of services has similarities with capability descriptions of different semantic web service frameworks [67] [86] [55]. These frameworks are quite generic and are targeted to be used in (extended by) different domains. These frameworks described other aspects to services such as their pre-conditions and effects, which are characteristics to be used during automated service composition and were not needed in myGrid. We believe that there exists no single appropriate data model for all domains. The data model we have in Feta is tailored towards our needs for bioinformatics services.

5.1 Contributors to Discovery Process

During the development of Feta we have come to realize that there are different types of users involved in the process of service publishing and discovery. These multiple stakeholders need to have different views over descriptions of services and domain knowledge. For the particular case of bioinformatics, there exist *service providers* who are responsible for publishing organizational information for a service, a textual description of its functionality, and a low-level interface specification. Additionally there are *annotators* who are users responsible for creating domain specific descriptions of services capabilities with respect to a commonly agreed capability model using simplified representation of the domain knowledge

Type of Reasoning / Time of Reasoning	Description Logic	RDF(S)	No Reasoning
Ontology Development	Feta View-Only Semantic-Rich	——	——
Publishing	Semantic-Rich	——	Feta View-Only
Discovery	Semantic-Rich	Feta	View-Only

Figure 5.2: The Lifecycle of Reasoning Employed During Discovery.

modelled in domain ontologies. *Ontologists* are the users who are responsible for creating detailed and formalized representations of the domain knowledge using ontology editors and DL reasoners. These components require their users to have a high level of know-how and expertise in knowledge-representation techniques. And finally there are *workflow builders* who are equipped with practical/usable tools that provide a simplified view of descriptions and classifications to aid discovery. We believe that taking these multiple stakeholders into account is important during the design and development of discovery frameworks. Previous discovery solutions in myGrid have also made such identifications among users, however no distinctions have been made among annotators and workflow designers. As stated in Chapter 1, service annotation and discovery can essentially be performed by the same user in myGrid. However we anticipate that the number of users performing discovery will be much larger than those who provide annotated descriptions for services. Furthermore these large number of service discoverers are more likely to be unfamiliar with myGrid's schema of services and would prefer even simpler user interfaces than Pedro. Based on this expectation we have provided separate interfaces for description generation and querying in Feta rather than using a single one for both activities (as done in previous approaches). Discoverers are provided with a simple interface that exposes a set of

pre-defined search requests over parts of the service schema that is expected to be most frequently searched over.

We were unable to observe any identification of multiple stakeholders in discovery in the most recent semantic service discovery research efforts. However we acknowledge that the identification of stake holders is a highly domain specific process.

5.2 myGrid Discovery Requirements and Feta

Table 5.1 displays our list of requirements and indicates the ones that are met by Feta. Feta meets requirement 1 and 2 by its user-oriented, operation-centric information model. What makes Feta's model user-oriented is its capture of the user's view of capabilities of workflow building blocks rather than the technology specific low-level invocation interface specifications. Requirement 4 is met by incorporation of domain classifications into descriptions and exploitation of the classification hierarchy during service search. The two forms of discovery supported by Feta (i.e. keyword based, and query based) meet requirements 5.a and 5.c. Feta also meets requirements 6.a and 6.c by providing a single interface to its search facilities from within Taverna.

Our objective was not to meet all requirements of service discovery in myGrid. (See items 2, 3.b, 3.c, 5.b, 6.b). Feta has been rather been developed to be complementary to other components in myGrid so that more of the requirements are satisfied. Within the following sections we will first speculate on how Feta can be augmented with existing myGrid components. We will then describe planned extensions to Feta in light of the unmet requirements.

5.3 Future work

5.3.1 Intended Use of Feta in myGrid

Feta has been demonstrated alongside the Taverna workbench at the 2004 Intelligent Systems in Molecular Biology (ISMB) conference. The ideas behind its design have obtained positive feedback from the bioinformatics community which currently has an increasing demand for a public bioinformatics registry. However, Feta itself is not intended to act as a registry, it is rather a semantic search

Req. No.	Description	Feta
1	User-oriented, workflow centric discovery of different types of services (i.e. operations)	Met
2.a	Dynamic discovery of shim services	Not met
2.b	Dynamic and Proactive discovery of candidate successor services in a workflow context	Partially met
3.a	Information Model - Service Capabilities (Domain Specific description)	Met
3.b	Information Model - Non-functional Aspects	Not Met
3.c	Information Model - Non-restricted 3rd party Assertions	Not Met
4	Exploiting domain knowledge	Met
5.a	Discovery by keyword based over name/text description	Met
5.b	Discovery by browsing a service classification hierarchy	Not Met
5.c	Discovery by sending search requests based on the information model	Met
6.a	Deployment - Unified interface for search facilities	Met
6.b	Management of service provider, and third-party descriptions by their owners	Not Met
6.c	Accessibility from the workflow design context	Met

Table 5.1: The Addressing of Discovery Requirements by Feta.

facility that can be augmented with a service registry. We believe that service registries (e.g. UDDI), with their design towards domain independent and robust publishing/querying facilities and their extensibility features (e.g. tModels of UDDI Information model) provide an ideal complement to domain dependent search facilities like Feta. As an immediate future work we are planning to deploy Feta's search facility as a **web service** that operates over descriptions advertised in a **public catalogue** of bioinformatics services.

Currently Feta is not delivered as a service, which makes it only accessible by human users using its GUI plug-in to Taverna. Feta's accessibility from other systems will be increased by its deployment as a web service.

There are several candidates to act as the service catalogue, such as the public UDDI registries or the myGrid View component for storing Feta descriptions of services. Among the candidate catalogues the myGrid View stands out as the one that will meet more of myGrid discovery requirements (3.c, 6.b) with its support for 3rd party metadata publishing without imposing a model/schema for the structure of metadata. Furthermore the View component is a UDDI-compatible registry. Integrating Feta with the View eliminates any further effort to integrate Feta with standard UDDI registries.

5.3.2 Information Model Extensions

5.3.2.1 Service Non-Functional Properties

In its current form Feta's information model only addresses the capability related aspects of a service. Workflow designers also want to be able to search for services based on their non-functional aspects such as performance, reliability, temporal and geographic availability.

Even though integrated use of Feta with the View would allow 3rd party annotations to be made on services which may as well be on non-functional aspects we think it would be more useful to the users if the non-functional aspects were explicitly addressed by Feta's information model.

Currently the detailed expectations from a model of non-functional aspects are not clearly identified. We believe that these requirements will be articulated:

(1) As soon as a public bioinformatics service registry augmented with Feta is deployed. Deployment of such a registry will proliferate semantic discovery of services and hence draw the communities attention to different aspects of

service description and discovery including non-functional aspects.

(2) Once an analysis of provenance logs for services is done. Provenance logs of *in silico* experiments contain information that can contribute to a non-functional characterization of services in a workflow. Information in provenance logs can be aggregated and used in updating non-functional service descriptions. Hence an investigation of the structure and content of provenance logs would be useful for sketching the main elements of a non-functional service description model.

5.3.3 Supporting Different Forms of Discovery

5.3.3.1 Knowledge Driven Workflow Design

Currently the search facility of Feta is invoked upon explicit requests coming from users (i.e. workflow designers). As future work on Feta we would like to extend its Taverna client components so that they become the initiating parties of discovery instead of the human users. Such a dynamic discovery architecture is needed to meet the requirement on proactive discovery and the suggestion of potential successor services in a workflow (Requirement 2.c). Similar proactive discovery systems have been developed in other e-Science projects such as Geodise [24].

5.3.3.2 Discovery by Browsing

Another form of discovery required by workflow designers is Discovery by Browsing a classification of services (Requirement 5.b). The domain classification used within Feta contains different classifications of domain concepts that are ued in characterization of service parameters, tasks, methods and so forth. As part of our future work on Feta we will provide users with these multiple hierarchies of domain concepts (a task hierarchy, a method hierarchy) and the services that are described by use of those concepts.

5.3.4 GUI Extensions

In its current status Feta's GUI plug-in to Taverna workbench is a prototype. The usability of our components is of great importance to us. Therefore we are planning to enhance the GUI interface of our system to make it more user-friendly based on user-feedback given in the evaluation section of Chapter 4. Additionally

any extensions or changes to the data model that is of interest to the workflow designers will be reflected to the Query Panel of the GUI plug-in.

5.3.5 Building a Custom Annotator

The annotator component Pedro commonly used in all discovery frameworks has so far met expectations, by reflecting myGrid's schema of services as a user-interface and by allowing domain specific descriptions to be generated with respect to this schema. In the systems developed, Pedro has been subject to extensions to be able to generate descriptions in forms other than its default form XML. As part of the future work on discovery we plan to develop our custom annotator component that would generate RDF descriptions of services.

5.3.6 Managing Changes to the Domain Ontology

The domain classification used in Feta is generated by exporting of the OWL based myGrid domain ontology as a classification. This classification is later fed into Feta's architecture as a static file. Such a solution makes Feta vulnerable to changes to the ontology. Possible changes can be: (1) changes in the DL based ontology can cause the classification hierarchy to change while the numbers of concepts or their names stay the same; (2) Changes in the DL ontology can cause additions and removals of classes. Feta's search engine should be able to cope with multiple versions of the myGrid domain classification that can be used in annotations of service descriptions and should adapt its search facilities accordingly so that descriptions are queried over by use of the correct version of the ontology used in their annotation. Such a solution necessitates use of an ontology server, with versioning capabilities, within Feta.

5.3.7 Use of Feta Outside the Scope of myGrid

Feta is designed to be deployed in myGrid's bioinformatics setting. Therefore it is tightly integrated to its information model of services. Feta can be re-used in other possible e-Science domains as long as its information model is applicable. An example use of Feta is currently underway in a sister bioinformatics project named MOBY [59]. In this project services used in plant genome analysis are to be discovered by bioinformaticians. Hence the information model of Feta has

been found suitable and Feta will be used as a semantic search facility with just re-configuring the domain ontology used within its architecture.

If Feta's information model is to be amended, any change would have the potential to invalidate the canned queries and GUI-query interface of Feta. Similarly new mapping rules from XML to RDF descriptions would be necessary. We might have designed the mapping rules to be configurable to provide for genericity and reuse of our discovery framework in other domains. However we did not find much value in making configurable mappings between XML to RDF when this configurable generic approach could not be propagated to the schema-dependent canned queries and the GUI interface that presents them and their results.

5.3.8 Enhanced Discovery in myGrid: Going Beyond Feta

Considering the current status of Feta together with its planned integration with the View and its future extensions, some of the requirements are still left unmet. These are the ones related to dynamic discovery, and planning of shim services and discovery of workflows based not only on their internal sub-components but also based on their data and control flow. Both of these areas of discovery require extensive and long-term research which is expected to incorporate AI planning and inexact graph matching techniques into myGrid service discovery environment respectively. Research in these areas has started to be undertaken in myGrid. Any outcomes of these efforts that has effects on Feta will be analyzed and incorporated as needed.

The time-frame spent during development of Feta has witnessed a fast growth in the number of bioinformatics services from a dozen to nearly a thousand. The growth is continuing as more service providers make their resources and tools available as services. In such a setting, discovery is becoming increasingly important for users who are faced with the challenge of selecting appropriate services to be used in workflows. It is our belief that, upon its deployment, Feta will be used by a large user base and become an vital part of the Bioinformatics *in silico* experiment lifecycle, without which workflow design would be very time consuming and tedious.

Bibliography

[1] Distributed Management Task Force – Common Information Model (CIM). http://www.dmtf.org/standards/cim/.

[2] Globus Monitoring and Discovery System-3 MDS-3. http://www.globus.org/mds/mds30.html.

[3] North American Industry Classification System (NAICS). http://www.naics.com/.

[4] Sesame Open Source RDF Database. http://www.openrdf.org/.

[5] The Dublin Core Metadata Initiative. http://dublincore.org/.

[6] The United Nations Standard Products and Services Code (UNSPSC). http://www.unspsc.org/.

[7] *Universal Description Discovery and Integration (UDDI) Technical Whitepaper.*

[8] Importing the Semantic Web in UDDI. In C. Bussler, R. Hull, S. McIlraith, M. E. Orlowska, B. Pernici, and J. Yang, editors, *Proceedings of Web Services, E-Business, and the Semantic Web: Caise 2002 International Workshop, WES*, volume 2512 of *Lecture Notes in Computer Science*, 2002.

[9] A. S. Ali, O. F. Rana, R. Al-Ali, and D. W. Walker. UDDIe: An Extended Registry for Web Services. In *Proceedings of Workshop on Service Oriented Computing: Models, Architectures and Applications.* IEEE Computer Society Press, 2003.

[10] S. Andreozzi, M. Sgaravatto, and C. Vistoli. Sharing a conceptual model of grid resources and services. *In Proceedings of Computing in High Energy*

and Nuclear Physics (CHEP)2003 Conference, June 2003. Provided by the NASA Astrophysics Data System.

[11] A. Ankolekar, M. Burstein, J. R. Hobbs, O. Lassila, D. McDermott, D. Martin, S. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara. DAML-S: Web Service Description for the Semantic Web. In I. Horrocks and J. A. Hendler, editors, *First International Semantic Web Conference*, volume 2342 of *Lecture Notes in Computer Science*. Springer, June 2002.

[12] A. Barros, K. Duddy, M. Lawley, Z. Milosevic, K. Raymond, and A. Wood. Processes, Roles, and Events: UML Concepts for Enterprise Architecture. In *The Unified Modeling Language, Advancing the Standard, Third International Conference*, volume 1939 of *Lecture Notes in Computer Science*, pages 62–77, York, UK, Oct. 2000. Springer.

[13] S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. OilEd: a Reason-able Ontology Editor for the Semantic Web. In *Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence*, volume 2174, pages 396–408. Springer-Verlag, LNAI, 2001.

[14] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):35–43, 2001.

[15] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol (SOAP). W3c recommendation, World Wide Web Consortium, May 8th, 2000 2000.

[16] D. Brickley, R. Guha, and B. McBride. RDF Vocabulary Description Language 1.0: RDF Schema. Working draft, World Wide Web Consortium, January 2003.

[17] C. Bussler, D. Fensel, and A. Maedche. The Web Services Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2):113–137, Summer 2002.

[18] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. Inference Support in Jena. On-line documentation.

[19] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. Jena: Implementing the Semantic Web Recommendations. Technical report, HP Labs, December 2003.

[20] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. *Web Services Description Language (WSDL) 1.1.*

[21] M. Clark. UDDI - The Weather Report. Article, Web Services Architect, 2001.

[22] Condor. `http://www.cs.wisc.edu/condor/`.

[23] A. Cooke, W. Nutt, J. Magowan, P. Taylor, J. Leake, R. Byrom, L. Field, S. Hicks, M. Soni, A. Wilson, R. Cordenonsi, L. Cornwall, A. Djaoui, S. Fisher, N. Podhorszki, B. Coghlan, S. Kenny, D. OCallaghan, and J. Ryan. Relational Grid Monitoring Architecture (R-GMA). In *Proceedings of the UK e-Science All Hands Meeting*. EPSRC, September 2003.

[24] S. Cox, L. Chen, S. Campobasso, M. Duta, M. Eres, M. Giles, C. Goble, Z. Jiao, A. Keane, G. Pound, A. Roberts, N. Shadbolt, F. Tao, J. Wason, and F. Xu. Grid Enabled Optimisation and Design Search (GEODISE). In *UK e-Science All Hands Meeting 2002*, Sheffield, UK, September 2002.

[25] F. Curbera, D. Ehnebuske, and D. Rogers. Using WSDL in a UDDI Registry, Version 1.07. Best practice document, May 2002.

[26] K. Czajkowski, D. F. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe. Web Services Resource Framework WSRF. Technical report, Globus Alliance and IBM, March 5th, 2004 2005.

[27] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *Proceedings of, 10th International Symposium on High Performance Distributed Computing*, HPDC, pages 181–194, San Francisco, CA, USA, August 2001. IEEE Press.

[28] K. Decker, K. Sycara, and M. Williamson. Middle-agents for the internet. In *In Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 578–583. Morgan Kaufmann Publishers, 1997.

[29] S. Decker, J. Jannink, S. Melnik, P. Mitra, S. Staab, R. Studer, and G. Wiederhold. An Information Food Chain for Advanced Applications on the WWW. In *Proceedings of the 4th European Conference on Research and Advanced Technology for Digital Libraries*, pages 490–493. Springer-Verlag, 2000.

[30] The e-Science Core Programme. `http://www.escience-grid.org.uk/`.

[31] D. Fensel, V. R. Benjamins, E. Motta, and B. J. Wielinga. UPML: A framework for knowledge system reuse. In *IJCAI*, pages 16–23, 1999.

[32] D. Fensel, F. van Harmelen, I. Horrocks, D. McGuinness, and P. F. Patel-Schneider. OIL: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2):38–45, 2001.

[33] R. T. Fielding and R. N. Taylor. Principled Design of the Modern Web Architecture. *ACM Transactions on Internet Technology (TOIT)*, 2(2):115–150, 2002.

[34] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[35] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. Grid Services for Distributed System Integration. *IEEE Computer Magazine*, 35(6):37–46, June 2002.

[36] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *The International Journal of High Performance Computing Applications*, 15(3):200–222, Fall 2001.

[37] K. L. Garwood, C. Taylor, K. Runte, A. Brass, S. Oliver, and N. W. Paton. Pedro: A Configurable Data Entry Tool for XML. *Bioinformatics*, 2004. To appear.

[38] D. Gisolfi. Is Web services the reincarnation of CORBA? IBM Developerworks On-line library, July 2001.

[39] C. A. Goble, S. Pettifer, R. Stevens, and C. Greenhalgh. *Enhancing Services and Applications with Knowledge and Semantics (Chapter23) The Grid:*

Blueprint for a New Computing Infrastructure. Morgan Kaufman, 2nd edition, 2003.

[40] C. A. Goble, S. Pettifer, R. Stevens, and C. Greenhalgh. *Knowledge Integration: In silico Experiments in Bioinformatics in The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufman, 2nd edition, 2003.

[41] T. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[42] V. Haarslev and R. Mller. High performance reasoning with very large knowledge bases. In *International Workshop in Description Logics 2000 (DL2000)*, 2000.

[43] I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.

[44] I. Horrocks. DAML+OIL: a description logic for the semantic web. *Bull. of the IEEE Computer Society Technical Committee on Data Engineering*, 25(1):4–9, Mar. 2002.

[45] I. Horrocks, L. Li, D. Turi, and S. Bechhofer. The Instance Store: DL Reasoning with Large Numbers of Individuals. In *Proc. of the 2004 Description Logic Workshop (DL 2004)*, pages 31–40, 2004.

[46] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The Making of a Web Ontology Language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[47] D. Hull, R. Stevens, P. Lord, and C. Goble. Integrating bioinformatics resources using shims. Intelligent Systems for Molecular Biology (ISMB): Poster, 2004.

[48] T. Kawamura, J.-A. D. Blasio, T. Hasegawa, M. Paolucci, and K. Sycara. Public Deployment of Semantic Service Matchmaker with UDDI Business Registry. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *Proc. of the 2004 International Semantic Web Conference (ISWC 2004)*, number 3298 in Lecture Notes in Computer Science, pages 471–485. Springer, 2004.

[49] T. Kawamura, J. D. Blasio, T. Hasegawa, M. Paolucci, and K. Sycara. Preliminary Report of Public Experiment of Semantic Service Matchmaker with UDDI Business Registry. In M. Orlwoska, S. Weerawarana, and M. Papazoglu, editors, *Proceedings of First International Conference on Service Oriented Computing (ICSOC 2003)*, number 2910 in Lecture Notes in Computer Science, pages 208–224. Springer, 2003.

[50] M. Kifer and G. Lausen. F-logic: a Higher-Order Language for Reasoning about Objects, Inheritance, and Scheme. In *Proceedings of ACM SIGMOD Conference*, pages 134–46, June 1989.

[51] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of ACM*, 42:741–843, July 1995.

[52] H. Knublauch. An AI tool for the real world - knowledge modeling with protege. *Java World*, June 2003.
Walkthrough of Protege.

[53] L. Krauter, R. Buyya, and M. Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software - Practice and Experience*, 32(2):135–164, 2002.

[54] R. Lara. Semantics for Web Service Discovery and Composition. Presentation as part of KnowledgeWeb Project Meeting, 2004. Computer Science Department University of Manchester.

[55] R. Lara, H. Lausen, S. Arroyo, J. de Bruijn, and D. Fensel. Semantic Web Services: description requirements and current technologies. In *Proceedings of the International Workshop on Electronic Commerce, Agents, and Semantic Web Services. (ICEC 2003)*, 2003.

[56] L. Li and I. Horrocks. A Software Framework for Matchmaking Based on Semantic Web Technology. In *Proceedings of the Twelfth International World Wide Web Conference WWW*, pages 331–339. ACM, 2003.

[57] P. Lord, S. Bechhofer, M. D. Wilkinson, G. Schiltz, D. Gessler, D. Hull, C. Goble, and L. Stein. Applying Semantic Web Services to bioinformatics: Experiences gained, lessons learnt. In *International Semantic Web Conference*, 2004. Accepted For Publication.

[58] F. Manola, E. Miller, and B. McBride. RDF Primer. Working draft, World Wide Web Consortium, January 2003.

[59] Mark D. Wilkinson and Matthew Links. BioMOBY: An open source biological web services proposal. *Briefings in Bioinformatics*, 3(4):331–341, 2002.

[60] D. L. McGuinness, R. Fikes, L. A. Stein, and J. Hendler. DAML-ONT: An Ontology Language for the Semantic Web. *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential.*, 2002.

[61] D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language: Overview. Working draft, World Wide Web Consortium, March 2003.

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

[63] K. Michalickova, G. D. Bader, M. Dumontier, H. Lien, D. B. R. Isserlin, and C. W. Hogue. SeqHound: biological sequence and structure database as a platform for bioinformatics research. *BMC Bioinformatics*, 3(32), 2002.

[64] S. Miles, J. Papay, T. Payne, K. Decker, and L. Moreau. Towards a Protocol for the Attachment of Semantic Descriptions to Grid Services. page 10, Jan. 2004.

[65] L. Moreau, S. Miles, J. Papay, K. Decker, and T. Payne. Publishing Semantic Descriptions of Services. GGF9, 2003.

[66] E. Motta. An Overview of the OCML Modelling Language. In *Proceedings of 8 th Workshop on Knowledge Engineering: Methods and Languages KEML*, 1998.

[67] E. Motta, J. Domingue, L. Cabral, and M. Gaspari. IRS-II: A Framework and Infrastructure for Semantic Web Services. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *In Proceedings of the 2nd International Semantic Web Conference 2003 (ISWC 2003)*, number 2870 in Lecture Notes in Computer Science. Springer, 2003.

[68] Object Management Group. *The Common Object Request Brocker: Architecture and Specification*, revision 2.1 edition, July 1997.

[69] Object Management Group. *CORBA Naming Service Specification*, May 2000.

[70] Object Management Group(OMG). *Trading Object Service Specification*, May 2000.

[71] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 2004.

[72] J. OSullivan, D. Edmond, and A. t. Hofstede. Whats in a service?: Towards accurate description of non-functional service properties. *Distributed and Parallel Databases Journal Special issue on E-Services*, 12(2-3):1117–133, 2002.

[73] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara. Semantic Matching of Web Services Capabilities. In I. Horrocks and J. A. Hendler, editors, *First International Semantic Web Conference*, volume 2342 of *Lecture Notes in Computer Science*. Springer, JUNE 2002.

[74] E. Prud'hommeaux and B. Grosof. RDF Query Survey. Technical report, W3C, April 2004.

[75] R. Raman, M. Livny, and M. H. Solomon. Matchmaking: Distributed Resource Management for High Throughput Computing. In *In Proc. 7th IEEE Symposium on High Performance Distributed Computing*, pages 140–149, 1998.

[76] S. Ran. A model for web services discovery with QoS. *SIGecom Exch.*, 4(1):1–10, 2003.

[77] P. Rice, I. Longden, and A. Bleasby. EMBOSS: The European Molecular Biology Open Software Suite. *Trends in Genetics*, 16(6):276–277, 2000.

[78] M. Sabou. Learning Web Service Ontologies from the myGrid Service Collection. Given at Information Management Group Seminar Series, October 2004. Computer Science Department University of Manchester.

[79] M. Senger, P. Rice, and T. Oinn. Soaplab – a unified Sesame door to analysis tools. In *Proceedings of the UK e-Science programme All Hands Conference*, September 2003.

[80] S. Shukla and A. Deshpande. Tutorial: Ldap directory services - just another database application? In W. Chen, J. F. Naughton, and P. A. Bernstein, editors, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, volume 29, page 580. ACM, 2000.

[81] R. Stevens, C. Goble, and S. Bechhofer. Ontology-based Knowledge Representation for Bioinformatics. *Briefings in Bioinformatics*, 1(4):398–416, November 2000.

[82] R. Stevens, H. Tipney, C. Wroe, T. Oinn, M. Senger, P. Lord, C. Goble, A. Brass, and M. Tassabehji. Exploring Williams Beuren Syndrome Using ^{my}Grid. In *Bioinformatics*, volume 20, pages i303–310, 2004. Intelligent Systems for Molecular Biology (ISMB) 2004.

[83] R. D. Stevens, A. J. Robinson, and C. A. Goble. ^{my}Grid: Personalised bioinformatics on the information grid . *Bioinformatics*, 17 Suppl. 1:302–302, 2003. ISMB 2003.

[84] K. Sycara, S. Widoff, M. Klusch, and J. Lu. LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace. *In Autonomous Agents and Multi-Agent Systems*, 5:173–203, 2002.

[85] H. Tangmunarunkit, S. Decker, and C. Kesselman. Ontology-based Resource Matching in the Grid—The Grid meets the Semantic Web. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *Proceedings of the 2003 International Semantic Web Conference ISWC*, number 2870 in Lecture Notes in Computer Science. Springer, 2003.

[86] The OWL-S Services Coalition. *OWL-S Semantic Markup for Web Services.*

[87] A. Tsalgatidou and T. Pilioura. An Overview of Standards and Related Technology in Web Services. *Journal of Distributed and Parallel Databases*, 12:135–162, 2002.

[88] University of Brighton, Information Technology Research Institute . What you see is what you meant. `http://www.itri.brighton.ac.uk/projects/WYSIWYM/`.

[89] M. Uschold, M. King, S. Moralee, and Y. Zorgios. The Enterprise Ontology. *The Knowledge Engineering Review*, 13(1):31–89, 1998.

[90] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. Service Location Protocol (SLP). Official Internet Protocol Standard, Internet Engineering Task Force, Network Working Group, 1997.

[91] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller. METEOR-S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services. *Journal of Information Technology and Management*, 2004.

[92] W. Vogels. Web services are not distributed objects. *IEEE Internet Computing*, 7(6):59–66, 2003.

[93] World Wide Web Consortium (W3C). *Web Services Architecture*, February 2004.

[94] C. Wroe, C. Goble, M. Greenwood, P. Lord, S. Miles, J. Papay, T. Payne, and L. Moreau. Automating Experiments Using Semantic Data on a Bioinformatics Grid. *IEEE Intelligent Systems*, 19(1):48–55, 2004.

[95] C. Wroe, R. Stevens, C. Goble, A. Roberts, and M. Greenwood. A Suite of DAML+OIL Ontologies to Describe Bioinformatics Web Services and Data. *the International Journal of Cooperative Information Systems*, 12(2):597–624, 2003.

[96] S. Zanikolas and R. Sakellariou. A taxonomy of grid monitoring systems. *Journal of Future Generation Computer Systems*, 2004. To appear.

Appendix A

myGrid Service Schema v.2 as XSD

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema targetNamespace="pd"
xmlns="pd" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:uddi="urn:uddi-org:api_v2"
elementFormDefault="qualified"
attributeFormDefault="unqualified">

  <xs:element name="serviceDescriptions">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="serviceDescription"
        minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="serviceDescription">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="serviceName"
        type="xs:string" minOccurs="0"/>
        <xs:element ref="organisation"
```

```
        minOccurs="0"/>
        <xs:element name="author"
        type="xs:string" minOccurs="0"/>
        <xs:element name="locationURL"
        type="xs:anyURI" minOccurs="0"/>
        <xs:element name="interfaceWSDL"
        type="xs:anyURI" minOccurs="0"/>
        <xs:element name="serviceDescriptionText"
        type="xs:string" minOccurs="0"/>
        <xs:element ref="operations"
        minOccurs="0"/>
        <xs:element name="serviceType"
        minOccurs="0">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Soaplab service"/>
            <xs:enumeration value="WSDL service"/>
            <xs:enumeration value="Workflow service"/>
          </xs:restriction>
        </xs:simpleType>
        </xs:element>

      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="serviceOperation">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="operationName" type="xs:string"
        minOccurs="0"/>
        <xs:element name="portName" type="xs:string"
        minOccurs="0"/>
        <xs:element name="operationDescriptionText"
        type="xs:string" minOccurs="0"/>
        <xs:element ref="operationInputs"
```

```
        minOccurs="0"/>
        <xs:element ref="operationOutputs"
        minOccurs="0"/>
        <xs:element name="operationTask"
        type="xs:string" minOccurs="0"/>
        <xs:element name="operationResource"
        type="xs:string" minOccurs="0"/>
        <xs:element name="operationMethod"
        type="xs:string" minOccurs="0"/>
        <xs:element name="operationApplication"
        type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="parameter">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="parameterName"
        type="xs:string" minOccurs="0"/>
        <xs:element name="messageName"
        type="xs:string" minOccurs="0"/>
        <xs:element name="parameterDescription"
        type="xs:string" minOccurs="0"/>
        <xs:element name="defaultValue"
        type="xs:string" minOccurs="0"/>
        <xs:element name="isConfigurationParameter"
        type="xs:boolean" minOccurs="0"/>
        <xs:element name="semanticType"
        type="xs:string"  minOccurs="0"/>
        <xs:element name="XMLSchemaTypeName"
        type="xs:string"  minOccurs="0"/>
        <xs:element name="XMLSchemaURI"
        type="xs:anyURI"  minOccurs="0"/>
        <xs:element ref="formats" minOccurs="0"/>
        <xs:element name="transportDataType"
```

```
      type="xs:string"  minOccurs="0"/>
      <xs:element name="collectionSemanticType"
      type="xs:string"  minOccurs="0"/>
      <xs:element name="collectionFormat"
      type="xs:string"  minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="operations">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="serviceOperation"
      minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="operationInputs">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="parameter"
      minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="operationOutputs">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="parameter"
      minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="formats">
  <xs:complexType>
    <xs:sequence>
```

```
      <xs:element name="formatIdentifier" type="xs:string"
      minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="contacts">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="contact"
      minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="contact">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="contactType"
      type="xs:string" minOccurs="0"/>
      <xs:element name="personName"
      type="xs:string" minOccurs="0"/>
      <xs:element name="description"
      type="xs:string" minOccurs="0"/>
      <xs:element name="phone" type="xs:string"
      minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="email" type="xs:string"
      minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="address" type="xs:string"
      minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="organisation">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="authorizedName"
```

```
        type="xs:string" minOccurs="0"/>
        <xs:element name="organisationName"
        type="xs:string" minOccurs="0"/>
        <xs:element name="organisationDescriptionText"
        type="xs:string"/>
        <xs:element name="organisationKey"
        type="xs:string" minOccurs="0"/>
        <xs:element name="contacts"
        minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Appendix B

Extract of myGrid Domain Classification

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE rdf:RDF [
 <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
 <!ENTITY a 'http://www.w3.org/2000/01/rdf-schema#'>
]>
<rdf:RDF xmlns:rdf="&rdf;"
 xmlns:a="&a;">

<a:Class rdf:about="http://www.mygrid.org.uk/ontology#
                    bioinformatics_application">
<a:subClassOf
rdf:resource="http://www.mygrid.org.uk/ontology#
bioinformatics_concept"/>
</a:Class>
<a:Class rdf:about="http://www.mygrid.org.uk/ontology#
                    bioinformatics_algorithm">
<a:subClassOf
rdf:resource="http://www.mygrid.org.uk/ontology#
bioinformatics_concept"/>
</a:Class>
<a:Class rdf:about="http://www.mygrid.org.uk/ontology#
                    bioinformatics_concept"/>
```

```
<a:Class rdf:about="http://www.mygrid.org.uk/ontology#
                    bioinformatics_data">
<a:subClassOf
rdf:resource="http://www.mygrid.org.uk/ontology#
bioinformatics_concept"/>
</a:Class>
<a:Class rdf:about="http://www.mygrid.org.uk/ontology#
                    bioinformatics_database">
<a:subClassOf
rdf:resource="http://www.mygrid.org.uk/ontology#
bioinformatics_concept"/>
</a:Class>

<a:Class rdf:about="http://www.mygrid.org.uk/ontology#
                    Basic_Local_Alignment_Search_Tool">
<a:subClassOf
rdf:resource="http://www.mygrid.org.uk/ontology#
bioinformatics_application"/>
</a:Class>
<a:Class rdf:about="http://www.mygrid.org.uk/ontology#
                    DDBJ">
<a:subClassOf
rdf:resource="http://www.mygrid.org.uk/ontology#
bioinformatics_database"/>
</a:Class>
<a:Class rdf:about="http://www.mygrid.org.uk/ontology#
                    EMBOSS">
<a:subClassOf
rdf:resource="http://www.mygrid.org.uk/ontology#
bioinformatics_application"/>
</a:Class>
<a:Class rdf:about="http://www.mygrid.org.uk/ontology#
                    FSSP">
<a:subClassOf
rdf:resource="http://www.mygrid.org.uk/ontology#
```

```
bioinformatics_database"/>
</a:Class>
<a:Class rdf:about="http://www.mygrid.org.uk/ontology#
                    SWISS-PROT">
<a:subClassOf
rdf:resource="http://www.mygrid.org.uk/ontology#
protein_sequence_database"/>
</a:Class>

<a:Class rdf:about="http://www.mygrid.org.uk/ontology#
                    sequence">
<a:subClassOf
rdf:resource="http://www.mygrid.org.uk/ontology#
bioinformatics_data"/>
</a:Class>

<a:Class rdf:about="http://www.mygrid.org.uk/ontology#
                    biological_sequence">
<a:subClassOf
rdf:resource="http://www.mygrid.org.uk/ontology#
sequence"/>
</a:Class>
<a:Class rdf:about="http://www.mygrid.org.uk/ontology#
                    nucleotide_sequence">
<a:subClassOf
rdf:resource="http://www.mygrid.org.uk/ontology#
biological_sequence"/>
</a:Class>
<a:Class rdf:about="http://www.mygrid.org.uk/ontology#
                    genome_nucleotide_sequence">
<a:subClassOf
rdf:resource="http://www.mygrid.org.uk/ontology#
nucleotide_sequence"/>
</a:Class>
```

```
<a:Class rdf:about="http://www.mygrid.org.uk/ontology#
                    DNA_sequence">
<a:subClassOf
rdf:resource="http://www.mygrid.org.uk/ontology#
nucleotide_sequence"/>
<a:subClassOf
rdf:resource="http://www.mygrid.org.uk/ontology#
sequence"/>
</a:Class>
<a:Class rdf:about="http://www.mygrid.org.uk/ontology#
                    primer_sequence">
<a:subClassOf
rdf:resource="http://www.mygrid.org.uk/ontology#
nucleotide_sequence"/>
</a:Class>

<a:Class rdf:about="http://www.mygrid.org.uk/ontology#
                    protein_sequence">
<a:subClassOf
rdf:resource="http://www.mygrid.org.uk/ontology#
biological_sequence"/>
</a:Class>
<a:Class rdf:about="http://www.mygrid.org.uk/ontology#
                    reverse_primer_sequence">
<a:subClassOf
rdf:resource="http://www.mygrid.org.uk/ontology#
primer_sequence"/>
</a:Class>


<a:Class rdf:about="http://www.mygrid.org.uk/ontology#
                    sequence_alignment_algorithm">
<a:subClassOf
rdf:resource="http://www.mygrid.org.uk/ontology#
bioinformatics_algorithm"/>
```

```
</a:Class>

<a:Class rdf:about="http://www.mygrid.org.uk/ontology#
         Needleman_and_Wunsch_global_sequence_alignment_algorithm">
<a:subClassOf
rdf:resource="http://www.mygrid.org.uk/ontology#
sequence_alignment_algorithm"/>
</a:Class>

<a:Class rdf:about="http://www.mygrid.org.uk/ontology#
                    Smith-Waterman_sequence_alignment_algorithm">
<a:subClassOf
rdf:resource="http://www.mygrid.org.uk/ontology#
sequence_alignment_algorithm"/>
</a:Class>

<a:Class rdf:about="http://www.mygrid.org.uk/ontology#
                    word_match_sequence_alignment_algorithm">
<a:subClassOf
rdf:resource="http://www.mygrid.org.uk/ontology#
sequence_alignment_algorithm"/>
</a:Class>

</rdf:RDF>
```