Chapter 1

AGENTS AND THE GRID: SERVICE DISCOVERY

Luc Moreau[1], Michael Luck[1], Simon Miles[1], Juri Papay[1], Keith Decker[1,2], Terry Payne[1]

Department of Electronics and Computer Science[1]
University of Southampton
L.Moreau,mml,sm,jp,ksd,trp@ecs.soton.ac.uk

Department of Computer and Information Sciences[2]
University of Delaware
decker@cis.udel.edu

Abstract The Grid is a large-scale computer system, capable of coordinating resources that are not subject to centralised control, while using standard, open, general-purpose protocols and interfaces, and delivering non-trivial qualities of service. In this chapter, we argue that Grid applications very strongly suggest the use of agent-based computing, and we review key uses of agent technologies in Grids: *user agents*, able to customise and personalise data, *agent communication languages* offering a generic and portable communication medium, and *negotiation* allowing multiple distributed entities to reach service-level agreements. In the second part of the chapter, we focus on Grid service discovery, which we have identified as a prime candidate for the use of agent technologies: we show that Grid services need to be located via personalised, semantic-rich discovery processes, which must rely on arbitrary metadata about services that originates from both service providers and service users. We present UDDI-M^T, an extension to the standard UDDI service directory approach that supports the storage of such metadata via a tunnelling technique that ties the metadata store to the original UDDI directory. The outcome is a flexible service registry that is compatible with existing standards and also provides metadata-enhanced service discovery.

Keywords: Agents, Grid computing, service discovery, bioinformatics.

1. Introduction

The Grid [Foster and Kesselman, 1998], an open computing infrastructure that supports large-scale distributed scientific research and applications, has recently gained heightened and sustained interest from several communities. It provides a means of developing a variety of e-Science applications including the study of genetic diseases (such as that described later in this chapter), particle physics making use of the Large Hadron Collider facility at CERN [datagrid, 2001], engineering design optimisation [Cox et al., 2001], and combinatorial chemistry [Frey et al., 2003]. The underlying computing infrastructure also supports more general applications that involve large-scale information handling, knowledge management and service provision [Roure et al., 2003]. Initially geared towards high performance computing, Grid computing is now being recognised as the future model for service-oriented environments, within and across enterprises, facilitating the formation of collections of coordinated services, or virtual organisations [Foster et al., 2001].

Large systems are naturally viewed in terms of the services they offer, and consequently in terms of the entities providing or consuming services. Grid applications typically consist of a set of such services that may be spread across a geographically distributed environment, and selected from a dynamically changing pool of available services. This service-oriented perspective, in which services (and their availability) may come and go, and collections of services are combined to achieve more complex tasks, very strongly suggests the use of agent-based computing [Luck et al., 2003]. In this view, agents act on behalf of service providers, managing access to services and ensuring that contracts are fulfilled. They also act on behalf of service consumers, locating services, agreeing contracts, and receiving and presenting results. Agents are required to engage in interactions, to negotiate, and to make pro-active run-time decisions while responding to changing circumstances.

In this chapter, we discuss the issues of agent-based Grid computing in the context of the *myGrid* project, and then focus in more detail on the specific issues involved in service discovery, which we identify as a prime candidate for use of agent technologies. We begin by describing myGrid, which seeks to provide Grid middleware for bioinformatics (Section 1.2), and then move to a general discussion of the role and use of agents in Grid computing for bioinformatics (Section 1.3). Section 1.4 addresses the use of agents for one area of Grid computing in more detail, namely service discovery. We review the current technologies and introduce in Section 1.5 a new service directory mechanism, UDDI-M^T, which augments the functionality of UDDI, the *de facto* standard directory for Web Services, with a metadata facility to better customise the publishing and discovering of services. As UDDI already offers a complex interface (for example, allowing searches on business categories and service names), UDDI-M^T uses a *tunnelling technique* for dispatching regular UDDI requests to a UDDI service, and intercepting UDDI-M^T metadata specific requests. Finally, we present conclusions and discuss further work.

2. The Grid and Bioinformatics

The Grid. The Grid is a large-scale computer system capable of coordinating resources that are not subject to centralised control, and which uses standard, open, general-purpose protocols and interfaces, delivering non-trivial qualities of service [Foster, 2002]. As part of the endeavour to define the Grid, a service-oriented approach has been adopted by which computational resources, storage resources, networks, programs and databases are all represented by services [Foster et al., 2002]. In this context, a service is a network-enabled entity capable of encapsulating diverse implementations behind a common interface.

The service-oriented aproach allows the composition of services into *workflows* in order to form more sophisticated services. Workflows are used for modelling the coordination between services, with each step in a workflow corresponding to an environment-dependent decision that must be made by some computational process.

In the e-Business community, a service-oriented architecture is also being adopted in the form of Web Services, which have emerged as a set of open standards, defined by the World Wide Web consortium and OASIS, and ubiquitously supported by IT vendors and users. Web Services rely on the XML syntactic framework, SOAP for exchanging messages, the WSDL interface definition language [wsdl, 2001], and the UDDI service directory [uddi, 2001].

Against this background, the Grid community has extended Web Services to support resource management required by Grid computing. This effort has resulted in the Open Grid Service Architecture (OGSA), a Grid architecture standardised by the Global Grid Forum, that defines a *Grid Service* as a Web Service providing a set of well-defined interfaces, following specific conventions [Foster et al., 2002]. In particular, Grid Services have some support for lifecycle management, and a conventional mechanism for discovery using service data elements (a service-specific type of metadata).

Bioinformatics. myGrid (`www.mygrid.org.uk`) is a pilot project funded by the UK e-Science programme to develop Grid middleware in a biological sciences context [Moreau et al., 2003]. To illustrate the functionality of Grid-based bioinformatics, myGrid has adopted an application that helps scientists study *Graves Disease*, a hormonal disorder caused by over-stimulation of the thyrotrophin receptor by thyroid-stimulating autoantibodies secreted by lymphocytes of the immune system. The Graves Disease application follows an *in-silico* experimental process typical of bioinformatics. In this process, the scientist: *(i)* attempts to discover information about candidate genes; *(ii)* makes an educated guess of the gene involved in the disease; and *(iii)* designs an experiment to be realised in the laboratory in order to validate the guess.

This *in-silico* experiment operates over the Grid, in which resources are geographically distributed and managed by multiple institutions, and the necessary tools, services and workflows are discovered and invoked dynamically. It is a *data-intensive* Grid, in which the complexity is in the data itself, the number of repositories and tools that need to be invoked in the computations, and the heterogeneity of the data, operations and tools.

In many resources, each record is analogous to an individual publication with not only raw data, but also additional annotations supplied by a small number of human experts (curators). Annotations are typically semi-structured text that may use keywords and controlled vocabularies, for parsing both by computers and by humans. Thus, in addition to a large number of data types, much of the valuable knowledge is locked into semi-structured text, under the premise that the scientist will read and interpret it.

In broad terms, myGrid follows common agent-oriented approaches in providing points at which automated processes can make decisions on what to do next depending on context. This manifests itself as automated service discovery, which we study in some detail in this chapter. First, however, we review the use of agent technologies in this context.

3. Agents in Bioinformatics Grids

Over the last few years, bioinformatics has undergone a rapid and substantial change. The key problem faced in this domain is the multitude, heterogeneity and variability of data, tools and technical literature available to bioscientists. Although there are several well-known and highly regarded databases, they are not exhaustive, and new ones often appear with new and different data. Thus, any system intended for application to the bioinformatics domain should be able to cope with this dynamism and openness, and nothing addresses these concerns as comprehensively as the agent approach. Agents are flexible, autonomous components designed to satisfy overarching strategic goals, while at the same time being able to respond to the uncertainty inherent in the environment. On the one hand, agents provide an appropriate paradigm or abstraction for the design of scalable systems aimed at this kind of problem; on the other, the field of agent-based computing offers a set of technologies that may be used for particular purposes in certain aspects of the system, including personalisation, communication and negotiation. It is the latter aspect of agent technologies that we analyse in this context, and discuss below.

User Agent

The *user agent* (also known as a *personal agent* [Maes, 1994]) is an agent in the sense that it *represents* a user within the myGrid system. It maintains a model of the user's goals and preferences, and uses these to make decisions and (if necessary) coordinate with other agents on the user's behalf. This is a useful feature, especially during workflow enactment, when a workflow is being executed and a choice of services becomes available. The choice should not be made arbitrarily, but based on the priorities and circumstances of the particular user. For example, a user may have more trust in the accuracy of one service than in others. Instead of querying the user each time a particular service needs to be selected, the user agent can mediate the selection process based on prescribed preferences, or on prior experience. This adaptive behaviour is known as *personalisation*.

Another application of the user agent is as a contact point between services within the Grid and the user. By introducing an intermediary able to receive, for example, requests from services for the user to enter data or notifications about changes to remote databases, these messages can be delivered to the user only when the user is able and willing to receive them. Conversely, the user can delegate repetitive actions to the user agent, such as authenticating itself with a service before use.

Agent Communication Language

The idea of an *agent communication language* dates back to the DARPA Knowledge Sharing Effort, which led to the design of KQML (Knowledge Query and Manipulation Language) [Finin et al., 1997], and was later followed by the FIPA (Foundation for Intelligent Physical Agents) Agent Communication Language [FIPA, 1997].

In multi-agent systems, it is common practice to separate intention from content in communicative acts, abstracting and classifying the former according to Searle's speech act theory [Searle, 1969]. Thus, an agent's communications can be structured and classified according to a predefined set of message categorisations, usually referred to as *performatives*.

In seeking to integrate agent communication with standard Grid technologies, we have previously described how the idea of agent communication languages could be mapped onto the communication stack of Web Services. First, we focused only on the communication layer by encoding performatives and message contents in SOAP [Moreau, 2002]. Second, we used the WSDL language for describing agents and the performatives they support [Avila-Rosas et al., 2002]. The aim of this research was to expose agent capabilities as Web Services so that agents can publish their capabilities (and subsequently be discovered) in a UDDI registry. The approach turns out to be promising, because it offers a declarative communication semantics, which promotes interoperability, openness, dynamic discovery and reuse of agents. It also opens the agent world to the Web Services community, helping in the design of more complex interactions.

Negotiation Broker

Service users and service providers typically have different criteria regarding the quality and content of services, but can resolve the differences through the use of negotiation. In Grid computing, one area in which negotiation can be particularly useful is in *notification* support. The providers of various services may want to send out notifications concerning improvements to tools, changes to databases or updates reflecting the state of enacted workflows, and so on. Other services or agents might want to register to receive a subset of these notifications. For stability, we consider asynchronous messages, and manage their distribution using a *notification service*.

The subjects (quantitative and qualitative) over which negotiation is undertaken could include the following forms of *quality of service*: the cost of receiving the notification; the topic (event category) of the notification; the frequency at which notifications are received; the generality of the change described by the notifications; and the format and accuracy of information contained in the notification message. These items, and many more, provide a metric for the quality of service.

An essential requirement for the smooth operation of any distributed system is that the consumer's demands of the service are met by the service providers. However, if these demands are not exactly met for some reason, the consumer may choose to *negotiate* with the publisher to find the next best quality of service that the publisher can provide. For example, the subscriber may require notifications weekly, whereas the publisher may only wish to provide them daily or fortnightly. In this case, the subscriber must choose between the available options or may decide not to subscribe at all, depending on their particular priorities. Alternatively, the publisher may be able to exceed the quality of service in several ways in which the subscriber may be unaware, and which could also lead to negotiation.

As the notification service must provide notification support for a potentially large and varying number of consumers, it should not change the contract covering the quality of service based solely on the results of negotiation between a single consumer and a publisher. Therefore, the notification service should have some control over the quality of service agreed upon. There are also other reasons why the notification service may usefully limit the interaction between the publisher and consumer, such as limiting one party's knowledge of the other for reasons of privacy or anonymity.

The *quality of service broker* described in [Lawley et al., 2003] is an agent conceptually contained within a notification service. This agent negotiates on behalf of each consumer wishing to receive notifications of a specified quality, and then produces a final proposal to both the consumer and the producer. It can negotiate with any of the publishers known to the notification service, and can also set boundaries on the agreed quality of service so that it is acceptable to the notification service.

4. Agent-Based Service Discovery for Grid Computing

Service discovery is a critical element in large scale, open distributed systems such as the Grid, as it facilitates the dynamic identification of resources abstracted as services. Providers may adopt various ways of describing their services, such as access polices or contract negotiation details. However, many resource consumers also impose their own selection policies on the services they prefer to use, such as derived quality of service, reputation metrics and. Consequently, both providers and consumers need to be able to locally manage and augment service descriptions with additional information, i.e. *metadata*.

The problem of service discovery is compounded by the plethora of different types of available service directories. Such services may include: *public* directories such as UDDI servers hosted by IBM or Microsoft; *specialised* directories such as the I3C bioinformatics service directory; *provider-specific* directories such as that of all the services hosted by a research institute; or even *local* directories such as the catalogue of all the services hosted by a laboratory for its own users. However, access to different types of service directory may require different protocols and query formats, with heterogeneous response formats.

Against this background, we have identified some key requirements that can enhance the process of service discovery by making the discovery process *personalised* to the user.

1 Users (not just service providers) should be able to attach metadata to service descriptions registered in service directories.

2 Users cannot be expected to systematically query all service directories in a discovery process. Instead, federating a selected set of service directories should provide a single point of access for the discovery process.

3 Users should be able to provide a semantic description of the task they want to locate, and the discovery should match the requirements against semantic descriptions of published services.

We will refer to the first two techniques as *syntactic*, whereas the third is *semantic*. Semantic descriptions can be used to specify what a service does in terms of the application domain (such as bioinformatics). Semantic techniques can then be applied to broaden or refine the list of services returned on discovery, based on the semantic descriptions and expert knowledge encoded in ontologies. For instance, queries for services of a general semantic type, such as sequence alignment tool, may also discover services described by a more specific type (sub-concept), such as BLAST tool. In the rest of this chapter, we shall only discuss the first technique, which allows users to attach metadata to published service instances. We refer the interested reader to [Lord et al., 2003] for a discussion of a semantic approach to service discovery.

Service Discovery Technologies

Service discovery has always played a crucial role in the evolution and deployment of distributed systems. Early distributed systems comprised collections of components (e.g. client-server or object-oriented) that were implicitly linked through function names, or linked through TCP/IP-based host and port addresses. The introduction of federated domain name servers (DNS) simplified and abstracted the use of numerical addresses by providing a registry-based mechanism for locating hosts. JINI [Arnold et al., 1999] used a similar approach as part of its Java-based distributed infrastructure. In this system, classes expose and publish their interfaces as *proxy objects* with the JINI discovery service. By searching for a given class-name, matching proxy objects can then be retrieved and invoked, which in turn call the remote service. While providing a mechanism by which services can easily be added, removed or replaced within a system, this approach is based on an assumption that there is a shared agreement about what a given service type is called (i.e. its class name) and that there was an agreed and well defined interface. Other distributed technologies support similar principles, including DCOM and Corba.

Web Services relax several assumptions of the JINI model. Unlike JINI, Web Services do not form well-defined class hierarchies, so it is difficult to locate services through class labels. To solve this problem, the UDDI service directory [uddi, 2001] was introduced, to register both service and provider specific information. The UDDI registry can be searched through a list of service descriptions, but there is little support provided for searches based on the service's signature or user-defined data.

UDDI supports the *tModel* (Technical Model) construct, which essentially serves two purposes: it provides a namespace for a taxonomy, and a proxy for a technical specification that lives outside the registry. tModels represent a powerful but limited mechanism for augmenting service registrations with metadata; their expressiveness was demonstrated by encoding properties from the DAML-Services (DAML-S) ontology [Ankolekar et al., 2002] within UDDI records [Paolucci et al., 2002].

However, before tModels may be used, they need to be registered with a UDDI server and hence be unique. While this is suitable for mapping well defined specifications to tModels, it is inappropriate for specifying large numbers of locally used metadata attributes (such as a set of attributes that may be shared by a single organisation or domain). The UDDI V3 specification attempts to amend this oversight by defining a specific tModel, *general_keywords*, to allow simple unregistered key-value pairs to be attached to a UDDI entity. However, this solution is oriented towards metadata supplied by the service *providers*, not users, and allows only simple textual metadata as opposed to more complex structures. An alternative approach to storing explicit, personalised, and possibly dynamic metadata that is associated with a service description is required to address these deficiencies.

In contrast to UDDI, the discovery services used by agent-based systems are typically designed to provide capability-based search, but provide little support for metadata-based search. These service registries index and search for registered services based on capability descriptions (or abstract descriptions of the service and its interface), rather than provider descriptions. Agents typically achieve their goals by identifying the types of services or tasks that need to be assembled together, and by delegating these tasks to those agents that provide the corresponding services. Such services are normally located by contacting one of several different *middle agents* [Decker et al., 1997]. Indeed, some middle agents have been developed for the adoption of a semantically-rich capability description language such as LARKS [Sycara and Klusch, 2001] and DAML-S [Ankolekar et al., 2002] to facilitate the matching of semantically equivalent, interoperable services, despite the fact that labels or syntactic constructs in the returned interface definition may not exactly match those in the query.

Interestingly, however, many Grid projects require large numbers of service and domain specific metadata. This might include, for example:

- perceived reputation of the service, which is critical to build a network of trusted services in an open environment;
- perceived reliability of the service, which has more value if it is provided by a third party, and not by the service provider itself;
- perceived quality of service by taking into account external factors, such as network connectivity, bandwidth, latency etc.;
- price for accessing a service (the user's institution may have negotiated a local price to access a resource, such as ACM or IEEE digital libraries); and
- ontological descriptions of a service, which may differ if there are multiple ontologies or interpretations of a service. While we may imagine that a whole scientific community shares a common ontology, the very nature of undertaking research necessarily entails that ontology revisions will be created by those who undertake this research, and who will therefore want to use them in order to characterise services within their refined ontologies.

Notions of services

One of the necessary elements to tackle in building a service directory relates to the differing notions of what a *service* actually is. Even within a unified model such as UDDI, the term is used in subtly different ways. In its most abstract form, the term *service* has two complementary meanings. For specifications driven mostly by the traditional distributed computing community (UDDI, WSDL), *service* tends to indicate a physical computing *entity* or entities that present some well-specified interface at particular physical endpoints. For specifications driven by the agents or more general AI community (DAML-S [Ankolekar et al., 2002]), *service* tends to indicate a *process* by which one may achieve a goal. These two viewpoints have significant overlap — an extremely common case in specification examples and in real implementations is one where a physical computing entity presents a well-specified interface which, in turn, enacts a process that achieves a goal. However, there are also situations that are harder to reconcile at this very high level of abstraction. First, from an agent perspective, a *service* could very well represent a large workflow quite explicitly spanning multiple physical computing entities (for example, composite DAML-S services), whereas a single service from the UDDI perspective may encompass many processes, each of which achieves different goals.

Not surprisingly, since the computational representation of semantic information has been a subject of study in AI for years, the most expressive models for semantic service description tend to be built around the agent-style service-as-process concept. While WSDL studiously avoids semantic information, UDDI does allow the assignment of predefined categorical values to both Business Services and Technical Services. Note that such categories are tied into the view of service-as-endpoint (e.g., geographical location) type of business, etc.

DAML-S attempts a full description of a service from the point of view that it is a process that can be enacted to achieve a goal. A full DAML-S service description incorporates three component perspectives: an abstract description of the service from the AI planning view (based on inputs, outputs, preconditions, and effects of a service — the service profile); the workflow view of the more primitive services needed to accomplish a complex goal (the service process); the mapping of the atomic parts of this workflow to their concrete WSDL descriptions (the service grounding). At its most complex, the DAML-S process view may be nested and include an explicit control model in order to monitor, alter, and possibly terminate the execution of a non-atomic service. Such models are analogous to emerging Web Service workflow proposals, such as WSFL and BPEL [Leyman, 2001, Curbera et al., 2002] and their associated standards, but this is beyond the scope of this chapter.

Service-Oriented Computing through Agent Technology

Service oriented computing, as described above in the context of the Grid, fits very well with agent technologies on the one hand, and the agent paradigm on the other. First, agent technologies of *middle agents* such as matchmakers and brokers can be used to address the problem of service discovery based on capability descriptions.

More generally, however, the paradigm of agent-based computing offers a way to view complex systems, with the area of agent-oriented software engineering [Wooldridge et al., 2000, Miles et al., 2000] providing ways of modelling and engineering such systems. These approaches make use of the *agent metaphor* to allow developers to reason about sophisticated behaviour in complex distributed systems, while avoiding explicit complexity in system design. In this view, the complexity arises out of the interactions of individual components at run-time rather than at design-time.

Clearly, however, distinct elements of the application must be identifiable as agents exhibiting flexible behaviour. Service directories matching the requirements given in the previous section are thus good candidates for an agent modelling approach for several reasons. First, they are federated, with annotations made in one registry being communicated to another with no guarantee of its inclusion in the latter: this is flexible social multi-agent behaviour. Such directories are *autonomous* in that they poll other registries according to some query, and *reactive* in that they may incorporate the results into their store, within the current environmental context provided both by a policy and by communication from other directories. Such flexible, autonomous, proactive, reactive behaviour demonstrates just those properties that characterise the agent approach.

Similarly, services surrounding service discovery can be viewed as agents. Agents can be present in the system as automatic publishers (or re-publishers) of services into multiple registries, as automated discoverers of services to be used in workflows, as personal agents adjusting service discovery to a user's preferences, and as automated executors that handle the invocation, composition and failure of services. Agents can also be used to regularly update metadata attached to a service description.

5. Architecture Design

The previous sections have identified Grid service discovery as a good candidate for deployment of agent technologies. Indeed, personalisation of content, flexible social behaviour through federation, and autonomy in the handling of requests, are all features of complex multi-agent systems. In this section, we describe a registry that can personalise content through a mechanism to attach metadata to service descriptions.

Since the types of personalised metadata that are required naturally vary greatly between individuals, organisations, and scientific user communities, an abstract and highly flexible representation is required. By regarding and implementing metadata items as triples that specify a relation between a subject and an object, arbitrary metadata can be described and queried via graph-based search criteria. This can be achieved through the use of RDF (the W3C Resource Description Framework) [rdf, 2001], which underpins the Semantic Web effort [Berners-Lee et al., 2001].

While UDDI is the acknowledged standard directory service mechanism for Web Services, it is limited in the kind of metadata that can be stored about services, the ways in which it can be queried, and who can annotate service descriptions with metadata. Our previous work, UDDI-M, was an early attempt to associate metadata with services and maintain soft state information [Foster et al., 2002], based on *leases* a la JINI. Both ideas have been reused in a service directory with quality of service information [ShaikhAli et al., 2003]. With UDDI-M^T, we take a further step by regarding and implementing metadata as triples, which gives us access to Semantic Web technologies such as RDF, and powerful query languages such as RDQL over a uniform representation of information. UDDI-M^T works in conjunction with a UDDI service to provide precisely these extra capabilities, and eventually support for personalised directory service federation and semantic service discovery.

In this section, we describe the principles underlying the architecture of UDDI-M^T, which was underpinned by the following requirements during the design:

1 UDDI-M^T should be compliant with the UDDI specification [uddi, 2001] and support future development in this direction.

2 Existing client applications and service providers should be able to make use of UDDI-M^T.

The key components of the architecture are depicted in Figure 1.1, where we see that UDDI-M^T is the point of contact for clients, used either for dispatching requests to UDDI or for processing them locally.

As far as implementation is concerned, UDDI-M^T was designed to be as generic as possible. First, all incoming requests are dispatched to the appropriate handler according to their type. Second, the UDDI-M^T backend is specified by an interface, which can be implemented in different ways: currently, we support a relational database and the Jena triple store [Jena, 2002].

This design assumes that all SOAP messages for the service directory issued by the client are routed to UDDI-M^T, which selectively filters them. This mechanism relies on the combination of the SOAP envelope and namespace contained in the message to dispatch the message to the appropriate handler, as specified by a configuration file. Messages with the UDDI namespace are

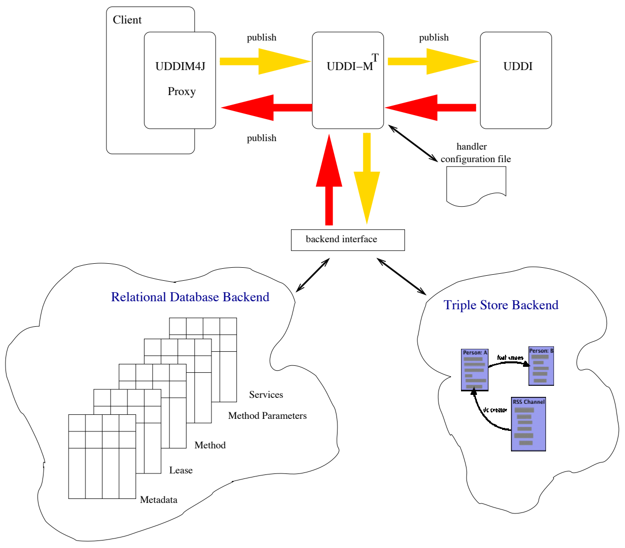


Figure 1.1. Architecture of UDDI-M^T

directly *tunnelled* to other UDDI registries, whereas messages with a UDDI-M^T namespace are handled locally.

All metadata-related information is stored in the UDDI-M^T backend. Its interface is implemented in two different ways. On the one hand, we use a relational database with five tables for metadata, leases, methods, method parameters and services. On the other hand, the same information is encoded by triples in a Jena triple store [Jena, 2002], for which three implementations are possible: an in-memory store, a relational database or the Berkeley Triple stores [Berkeley, 2002].

UDDI-M^T offers several extensions to UDDI. First, UDDI-M^T allows the association of metadata with services. Second, it supports a lease mechanism that requires services to renew their lease in order to maintain their registration in UDDI-M^T; such functionality is present in JINI [Arnold et al., 1999] and is also ubiquitous in the Open Grid Services Architecture [Foster et al., 2002]. Third, UDDI-M^T is able to extract the information contained in WSDL files describing the interface. Fourth, UDDI-M^T extends the query mechanism of UDDI to allow searches of all the extra information it accumulates about services. Fifth, in the specific case of the Jena backend, UDDI-M^T allows users to express queries in the RDQL-query language [Jena, 2002], offering homogeneous ways of traversing the metadata graph associated with services.

While UDDI is defined as a Web Service, a programmatic interface (UDDI4J [uddi4j, 2001]) is also available for Java: it provides a client-side proxy with an API implementing the UDDI functionality, which allows programmers to abstract away from the messaging layer. We have extended this proxy, by subclassing it, with additional UDDI-M^T functions for managing leases and metadata. Thus, we preserve clients' binary compatibility. Indeed, a UDDI proxy can be transparently substituted for a UDDI-M^T proxy, in existing clients, since the latter is a subclass of the former. Clients do not have to use the functionality provided by UDDI-M^T, they can use the existing namespace specification and the calls will be directly tunnelled to the underlying UDDI service.

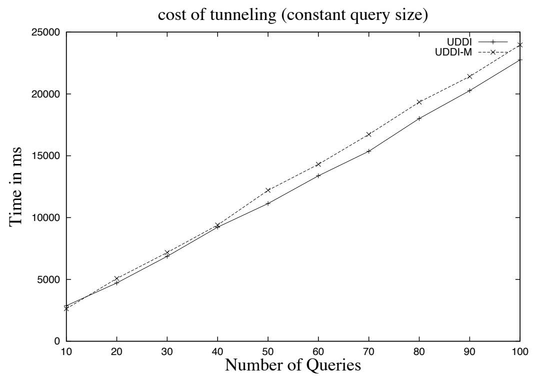
6. Performance Analysis

The Grid *checklist* [Foster, 2002] identifies "non-trivial qualities of service" as an essential feature of Grids. In taking this on board, the purpose of this section is to understand the impact of our design decisions on the performance of discovery. We focus our attention on two specific aspects. First, adopting the tunnelling technique reduces the implementation effort and allows us to maintain compatibility with evolving standards, but it comes at the price of SOAP-message forwarding; in the first part of this section, we analyse the cost of tunnelling. Second, the use of metadata in a service directory allows us to reduce the computational load on clients, while performing more selective and computationally intensive queries at the server side; in the second part of this section, we analyse the cost of metadata querying, and see how the use of the RDQL language, offering extended expressiveness to the user, impacts on the querying cost. A more detailed analysis can be foung in [Miles et al., 2003].

Tunnelling Cost. Our hypothesis is that *the overhead introduced by the tunnelling technique is acceptable*. In order to evaluate such a hypothesis, we have set up the following experimental framework.

A UDDI-M^T service and its associated UDDI service are hosted in a Tomcat server. A client uses a UDDI4J proxy successively configured to use UDDI and UDDI-M^T. In order to avoid the cost of networking, both the client and services are run on a same machine, and communications take place through the "localhost" network device. We issue a UDDI-query that searches for a service with a specific name, for which a single instance has been registered. Figure 1.2 shows the overhead introduced by UDDI-M^T, which tunnels the request to UDDI. The tests were run on a Pentium 4, 1.5GHz, with 512Mb, using Tomcat 4.0 and the Registry Server 1.0_02, in the Java Web Services Developer Pack (1.0_01). The data plotted were averaged over 10 runs. The tunnelling overhead is 7.2%.

We also evaluated the cost of tunnelling as the size of query results increases, but did not obtain any significant result, as the marginal tunnelling cost was noise compared to the querying cost.



UDDI	
Mean	189.9ms
Std Dev	24.9ms
UDDI-M	
Mean	203.7ms
Std Dev	24.8ms
Abs Overhead	13.8ms
Rel Overhead	7.2%

Figure 1.2. Overhead of Tunnelling

Metadata Querying Cost. Our second hypothesis is that *using a triple store as an internal representation mechanism for* UDDI-M^T *is practical, and the use of the* RDQL*-query language can reduce communication costs, and offload the client, by performing some server-side computation.* For these experiments the associated UDDI is not involved, resulting in a commensurate reduction in query times from the previous experiments.

In Figure 1.3, we show the costs of attaching a property value pair to a service already registered, which we call a *property write* operation, and of finding a service with a given property value pair, which we call a *property read* operation. We used the two different backends, a mySQL relational database and Jena with the Berkeley triple store for these experiments. For the Jena backend, we used the Jena API to find the service with given metadata, and we did not rely on the RDQL-query language. We plotted the results in Figure 1.3(a) using a logarithmic scale to differentiate the curves better. *Our purpose here is not to compare persistent storage technologies, but to understand the cost of metadata management. We can see that read operations for both backends and the write operation in the Jena store are very similar.* We explain the higher cost for the write operation with the SQL database by the cost of storing information on disk, which is probably not measured with the triple store.

In Figure 1.3(b), we used the RDQL-query language to search for a service satisfying 100 properties; 20 such services were found in the system. For convenience, we again plotted the Jena *read* line from Figure 1.3(a). We can see that the RDQL-query engine processing a complex query that checks 100 properties marginally outperforms our direct use of the triple store API, which itself behaved well compared to a relational backend.

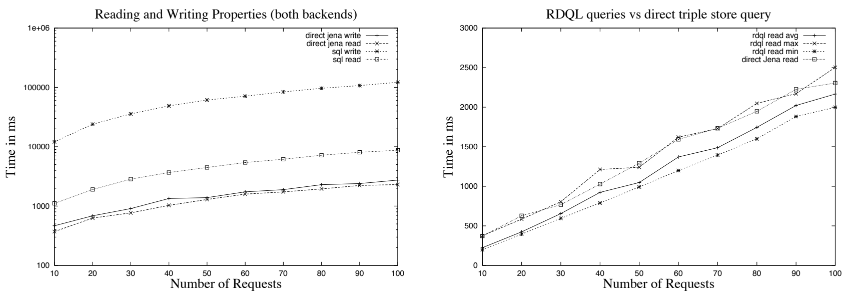


Figure 1.3. (a) Property Read and Write (b) RDQL Queries

7. Related Work

The notion of agents has recently become popular in the Grid community. Rana and Walker [Rana and Walker, 2000] advocate the use of the agent paradigm to integrate multiple information sources in problem-solving environments. Busetta *et al.* [Busetta et al., 2001] describe a Belief-Desire-Intention (BDI) agent architecture to simulate query optimisations in the Data Grid; their long term goal is to provide advanced and adaptable Grid services (of which query optimisation is one) based on agent technologies. Rana and Moreau [Rana and Moreau, 2000] review how agent techniques may be used to implement services at the computational Grid layer.

The DARPA CoABS (Control of Agent Based Systems) Grid [coabs, 2000] integrates various heterogeneous agent-based systems, mobile agent systems, object-based and legacy systems. It is based on JINI [Oaks and Wong, 2000] for its lookup service and Java RMI for inter-agent communication, and tests of scalability of the registration mechanism have been undertaken in [Kahn and Cicalese, 2001].

Without mentioning agents explicitly, Furmento, Newhouse and Darlington [Furmento et al., 2001] discuss another JINI-based technique for federating resources. Their long-term goal is the building of a computational economy for the Grid. Several other projects investigate this idea of a computational economy, according to which an economics framework regulates the supply and demand of resources. In particular, Nimrod/G [Buyya et al., 2001] is a resource broker capable of *budget-based* scheduling, giving users incentives to trade execution time for economic cost.

From the agent side, the community has been very active in devising high-level interaction protocols able to coordinate the activities of suppliers and consumers. Agents may *cooperate* in order to achieve a common goal, resulting in cooperative problem-solving which, sometimes, gives rise to adaptive behaviour. An alternative approach to this cooperation paradigm is the *market-based* model in which agents act as self-interested entities competing in a market, where goods such as computational resources are traded. Systems based on this paradigm have been shown to reach an overall equilibrium, in which resources are efficiently allocated [Clearwater, 1996, Kuwabara et al., 1996, Miller et al., 1996]. The market-based approach gives good results in particular when resources become scarce, and is a specific case of the more general type of interaction among self-interested agents, *negotiation* [Jennings et al., 1998]. As suggested earlier in this chapter, the key characteristics of negotiation are the presence of some form of conflict that must be resolved in a decentralised manner, by self-interested rational agents with incomplete information. Negotiation is the paradigm case of persuasion. It is a process by which agents come to a mutually acceptable agreement; apart from the work mentioned earlier, we are not aware of any of these techniques being applied in the context of the Grid.

Although the paradigm of agents has been used in the context of bioinformatics previously, this has not taken a Grid perspective. For instance, both [Decker et al., 2001] and [Bryson et al., 2002] used agent systems to federate data sources and tools in bioinformatics applications.

8. Conclusion and Future Work

In this chapter, we have discussed the roles and use of agents in Grid computing in general, but drilled down to explore service discovery in more detail. The examples of the use of agents that we have presented offer substantial new capabilities for Grid computing but still remain rather localised to some specific services. In particular, the full potential of agent technologies is yet to be exploited in future features of our service directory.

More specifically, the chapter describes the need for attaching metadata to services registered in service directories. This metadata describes functional and non-functional characteristics of services, and can be supplied by both publishers and consumers of a service. We have presented the architecture and the implementation of UDDI-M^T, an extension of UDDI, supporting metadata attachment and query. Our experimental evaluation has demonstrated the soundness of architectural design and implementation.

For the long term, agent-based computing also counts in its armoury a range of techniques for enabling individual components to collaborate with others, as well as for competing with others in the provision of services as may be found in bioinformatics. For example, the former aspects include issues in the construction of virtual organisations, whereby different services come together in some coherent whole subsystem for a particular purpose; and issues in the regulation of open societies of services through the use of norms and electronic institutions. The latter aspects, for example, include the possible use of sophisticated auction mechanisms, or electronic marketplaces, for obtaining the best services or resources at the least cost to the user. Additionally, whenever interactions take place between different agents, the issues of provenance, trust and reputation become important. Though some work has been done in this area, the focus on both agent-based computing and Grid computing has been limited, with the majority adopting the stance of assuming complete trust, and avoiding the issue; questions of deception and fraud in communication and interaction, of assurance and reputation, and of risk and confidence, are particularly significant, especially where interactions take place with new partners.

9. Acknowledgement

This research is funded in part by EPSRC myGrid project (reference GR/R67743/01). Thanks to myGrid colleagues: Matthew Addis, Nedim Alpdemir, Andy Brass, Rich Cawley, Neil Davis, David De Roure, Vijay Dialani, Alvaro Fernandes, Justin Ferris, Rob Gaizauskas, Kevin Glover, Carole Goble, Mark Greenwood, Chris Greenhalgh, Yikun Guo, Simon Harper, Clare Jennings, Ananth Krishna, Peter Li, Xiaojian Liu, Phillip Lord, Darren Marvin, Karon Mee, Arijit Mukherjee, Tom Oinn, Steve Oliver, Savas Parastiditis, Norman Paton, Simon Pearce, Stephen Pettifer, Milena V Radenkovic, Peter Rice, Angus Roberts, Alan Robinson, Tom Rodden, Martin Senger, Nick Sharman, Robert Stevens, Victor Tan, Brian Warboys, Paul Watson, Anil Wipat, Chris Wroe.

References

[Ankolekar et al., 2002] Ankolekar, Anupriya, Burstein, Mark, Hobbs, Jerry, Lassila, Ora, McDermott, Drew, Martin, David, McIlraith, Sheila, Narayanan, Srini, Paolucci, Massimo, Payne, Terry, and Sycara, Katia (2002). DAML-S: Web Service Description for the Semantic Web. In *First International Semantic Web Conference (ISWC) Proceedings*, pages 348–363.

[Arnold et al., 1999] Arnold, Ken, O'Sullivan, Bryan, Scheifler, Robert, Waldo, Jim, and Wollrath, Ann (1999). *The Jini Specification*. Sun Microsystems.

[Avila-Rosas et al., 2002] Avila-Rosas, Arturo, Moreau, Luc, Dialani, Vijay, Miles, Simon, and Liu, Xiaojian (2002). Agents for the Grid: A Comparison with Web Services (part II: Service Discovery). In *Workshop on Challenges in Open Agent Systems*, Bologna, Italy.

[Berkeley, 2002] Berkeley (2002). Berkeley DB. http://www.sleepycat.com/.

[Berners-Lee et al., 2001] Berners-Lee, Tim, Hendler, James, and Lassila, Ora (2001). The Semantic Web. *Scientific American*, 284(5):34–43.

[Bryson et al., 2002] Bryson, Kevin, Luck, Michael, Joy, Mike, and Jones, David (2002). Agent Interaction for Bioinformatics Data Management. *Applied Artificial Intelligence*.

[Busetta et al., 2001] Busetta, Paolo, Carman, Mark, Serafini, Luciano, Stockinger, Kurt, and Zini, Floriano (2001). Grid Query Optimisation in the Data Grid. Technical Report IRST 0109-01, Istituto Trentino di Cultura.

[Buyya et al., 2001] Buyya, Rajkumar, Giddy, Jonathan, and Abramson, David (2001). An economy grid architecture for service-oriented grid computing. In *10th IEEE International Heterogeneous Computing Workshop (HCW 2001), In conjunction with IPDPS 2001*, San Francisco, USA.

[Clearwater, 1996] Clearwater, Scott H., editor (1996). *Market-Based Control. A Paradigm for Distributed Resource Allocation*. World Scientific Publishing.

[coabs, 2000] coabs (2000). The DARPA CoABS Project: "Control of Agent Based Systems". `http://coabs.globalinfotek.com/`.

[Cox et al., 2001] Cox, Simon J., Fairman, Matthew J., Xue, Gang, Wason, Jasmin L., and Keane, Andy J. (2001). The Grid: Computational and Data Resource Sharing in Engineering Optimisation and Design Search. In *IEEE Proceedings of the 2001 ICPP Workshops*, pages 207–212, Valencia, Spain.

[Curbera et al., 2002] Curbera, Francisco, Goland, Yaron, Klein, Johannes, Leymann, Frank, Roller, Dieter, Thatte, Satish, and Weerawarana, Sanjiva (2002). Business process execution language for web services. http://www.ibm.com/developerworks/library/ws-bpel/.

[datagrid, 2001] datagrid (2001). The datagrid project. http://eu-datagrid.web.cern.ch/eu-datagrid/.

[Decker et al., 1997] Decker, Keith, Sycara, Katia, and Williamson, Mike (1997). Middle-Agents for the Internet. In *IJCAI97*.

[Decker et al., 2001] Decker, Keith, Zheng, X., and Schmidt, C. (2001). A Multi-Agent System for Automated Genetic Annotation. In *The fifth ACM International Conference on Autonomous Agents*, Montreal, Canada.

[Finin et al., 1997] Finin, Tim, Labrou, Yannis, and Mayfield, James (1997). *Software Agents, J. Bradshaw, Ed.*, chapter KQML as an Agent Communication Language. MIT Press.

[FIPA, 1997] FIPA (1997). FIPA: Foundation for Intelligent Physical Agents. `http://www.fipa.org/`.

[Foster, 2002] Foster, Ian (2002). What is the grid? a three point checklist. http://www-fp.mcs.anl.gov/ foster/.

[Foster and Kesselman, 1998] Foster, Ian and Kesselman, Carl, editors (1998). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufman Publishers.

[Foster et al., 2002] Foster, Ian, Kesselman, Carl, Nick, Jeffrey M., and Tuecke, Steven (2002). The Physiology of the Grid — An Open Grid Services Architecture for Distributed Systems Integration. Technical report, Argonne National Laboratory.

[Foster et al., 2001] Foster, Ian, Kesselman, Carl, and Tuecke, Steve (2001). The Anatomy of the Grid. Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*.

[Frey et al., 2003] Frey, Jeremy G., Bradley, Mark, Essex, Jonathan W., Hursthouse, Michael B., Lewis, Susan M., Luck, Michael M., Moreau, Luc, Roure, Dave C. De, Surridge, Mike, and Welsh, Alan (2003). *Grid Computing — Making the Global Infrastructure a Reality (Editors: Fran Berman, Geoffrey Fox and Tony Hey)*, chapter Combinatorial chemistry and the Grid, pages 945–962. Wiley Series in Communications Networking and Distributed Systems. John Wiley and Sons, Chichester, England.

[Furmento et al., 2001] Furmento, Nathalie, Newhouse, Steven, and Darlington, John (2001). Building Computational Communities from Federated Resources_ In *Proceedings of the 7th International Euro-Par Conference (Euro-Par 2001)*, pages 855–863, Manchester, UK.

[Jena, 2002] Jena (2002). Jena semantic web toolkit. `http://www.hpl.hp.com/semweb/jena.htm`.

[Jennings et al., 1998] Jennings, Nicholas R., Sycara, Katia, and Wooldridge, Michael (1998). A Roadmap of Agent Research and Development. *Int. Journal of Autonomous Agents and Multi-Agent Systems*, 1(1):7–38.

[Kahn and Cicalese, 2001] Kahn, Martha L. and Cicalese, Cynthia Della Torre (2001). CoABS Grid Scalability Experiments. In *Second International Workshop on Infrastructure for Scalable Multi-Agent systems at Autonomous Agents*, Montreal, Canada.

[Kuwabara et al., 1996] Kuwabara, Kazuhiro, Ishida, Toru, Nishibe, Yoshiyasu, and Suda, Tatsuya (1996). An Equilibratory Market-Based Approach for Distributed Resource Allocation and Its Applications to Communication Network Control. In *Market-Based Control. A Paradigm for Distributed Resource Allocation*, pages 53–73. World Scientific Publishing.

[Lawley et al., 2003] Lawley, Richard, Decker, Keith, Luck, Mike, Payne, Terry, and Moreau, Luc (2003). Automated negotiation for grid notification services. In *Ninth International Europar Conference (EURO-PAR'03)*, Lecture Notes in Computer Science, Klagenfurt, Austria. Springer-Verlag.

[Leyman, 2001] Leyman, Frank (2001). Web Services Flow Language (WSFL). Technical report, IBM.

[Lord et al., 2003] Lord, Phillip, Wroe, Chris, Stevens, Robert, Goble, Carole, Miles, Simon, Moreau, Luc, Decker, Keith, Payne, Terry, and Papay, Juri (2003). Semantic and Personalised Service Discovery. In Cheung, W. K. and Ye, Y., editors, *Proceedings of Workshop on Knowledge Grid and Grid Intelligence (KGGI'03), in conjunction with 2003 IEEE/WIC International Conference on Web Intelligence/Intelligent Agent Technology*, pages 100–107, Halifax, Canada. Department of Mathematics and Computing Science, Saint Mary's University, Halifax, Nova Scotia, Canada.

[Luck et al., 2003] Luck, Michael, McBurney, Peter, and Preist, Chris (2003). *Agent Technology: Enabling Next Generation Computing*. AgentLink.

[Maes, 1994] Maes, Pattie (1994). Agents that Reduce Work and Information Overload. *Communications of the ACM*, 37(7):31–40.

[Miles et al., 2000] Miles, Simon, Joy, Mike, and Luck, Michael (2000). Designing agent-oriented systems by analysing agent interactions. In Ciancarini, P. and Wooldridge, M. J., editors, *Proceedings of Agent-Oriented Software Engineering 2000 (AOSE 2000)*, pages 171–184.

[Miles et al., 2003] Miles, Simon, Papay, Juri, Dialani, Vijay, Luck, Michael, Decker, Keith, Payne, Terry, and Moreau, Luc (2003). Personalised grid service discovery. *IEE Proceedings Software: Special Issue on Performance Engineering*, 150(4):252–256.

[Miller et al., 1996] Miller, Mark S., Krieger, David, Hardy, Norman, Hibbert, Chris, and Tribble, E. Dean (1996). An Automated Auction in ATM Network Bandwidth. In *Market-Based Control. A Paradigm for Distributed Resource Allocation*, pages 96–125. World Scientific Publishing.

[Moreau, 2002] Moreau, Luc (2002). Agents for the Grid: A Comparison with Web Services (Part 1: the transport layer). In Bal, Henri E., Lohr, Klaus-Peter, and Reinefeld, Alexander, editors, *Second IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2002)*, pages 220–228, Berlin, Germany. IEEE Computer Society.

[Moreau et al., 2003] Moreau, Luc, Miles, Simon, Goble, Carole, Greenwood, Mark, Dialani, Vijay, Addis, Matthew, Alpdemir, Nedim, Cawley, Rich, Roure, David De, Ferris, Justin, Gaizauskas, Rob, Glover, Kevin, Greenhalgh, Chris, Li, Peter, Liu, Xiaojian, Lord, Phillip, Luck, Michael, Marvin, Darren, Oinn, Tom, Paton, Norman, Pettifer, Stephen, Radenkovic, Milena V, Roberts, Angus, Robinson, Alan, Rodden, Tom, Senger, Martin, Sharman, Nick, Stevens, Robert, Warboys, Brian, Wipat, Anil, and Wroe, Chris (2003). On the Use of Agents in a BioInformatics Grid. In Lee, Sangsan, Sekguchi, Satoshi, Matsuoka, Satoshi, and Sato, Mitsuhisa, editors, *Proceedings of the Third IEEE/ACM CCGRID'2003 Workshop on Agent Based Cluster and Grid Computing*, pages 653–661, Tokyo, Japan.

[Oaks and Wong, 2000] Oaks, Scott and Wong, Henry (2000). *Jini In a Nutshell*. O'Reilly.

[Paolucci et al., 2002] Paolucci, Massimo, Kawamura, Takahiro, Payne, Terry R., and Sycara, Katia (2002). Importing the Semantic Web in UDDI. In *Web Services, E-Business and Semantic Web Workshop*.

[Rana and Moreau, 2000] Rana, Omer F. and Moreau, Luc (2000). Issues in Building Agent based Computational Grids. In *Third Workshop of the UK Special Interest Group on Multi-Agent Systems (UKMAS'2000)*, Oxford, UK.

[Rana and Walker, 2000] Rana, Omer F. and Walker, David W. (2000). 'The Agent Grid': Agent-Based Resource Integration in PSEs. In *Proceedings of the 16th IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation*, Lausanne, Switzerland.

[rdf, 2001] rdf (2001). Resource Description Framework (RDF). `http://www.w3.org/RDF/`.

[Roure et al., 2003] Roure, David De, Jennings, Nicholas, and Shadbolt, Nigel (2003). The Semantic Grid: A Future e-Science Infrastructure. In Berman, F., Fox, G., and Hey, A. J. G., editors, *Grid Computing: Making the Global Infrastructure a Reality*, pages 437–470. Wiley.

[Searle, 1969] Searle, John (1969). *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press.

[ShaikhAli et al., 2003] ShaikhAli, Ali, Rana, Omer F., Al-Ali, Rashid, and Walker, David W. (2003). UDDIe: An Extended Registry for Web Services. In *Workshop on Service Oriented Computing: Models, Architectures and Applications at SAINT Conference*. IEEE Computer Society Press.

[Sycara and Klusch, 2001] Sycara, Katia and Klusch, Mattheus (2001). Brokering and matchmaking for coordination of agent societies: A survey. In et al, Omicini, editor, *Coordination of Internet Agents*. Springer.

[uddi, 2001] uddi (2001). Universal Description, Discovery and Integration of Business of the Web. `www.uddi.org`.

[uddi4j, 2001] uddi4j (2001). UDDI4J Home Page. `www.uddi4j.org`.

[Wooldridge et al., 2000] Wooldridge, Michael, Jennings, Nicholas R., and Kinny, David (2000). The Gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3.

[wsdl, 2001] wsdl (2001). Web Services Description Language (WSDL). `http://www.w3.org/TR/wsdl`.