

Engineering an Agent-Based Peer-To-Peer Resource Discovery System

Andrew Smithson and Luc Moreau*
{ans199,L.Moreau}@ecs.soton.ac.uk

Department of Electronics and Computer Science
University of Southampton
Southampton SO17 1BJ UK

Abstract. We have designed an agent-based peer-to-peer resource discovery system, which combines a set of original features. We distinguish synchronous and asynchronous searches, and structure them in terms of speech acts in an agent communication language. We rely on a distributed reference counting mechanism to detect the termination of asynchronous distributed searches. Ontologies are used to define resource descriptors in an extensible and open manner, as well as queries over such resources. A graphical user interface dynamically constructed from available resource descriptors is proposed. The system has been fully implemented in SoFAR, the Southampton Framework for Agent Research.

1 Introduction

Resource discovery is a critical activity in large scale distributed systems such as Grid environments [1]. However, these environments tend to be constituted by the confederation of multiple domains [2] or the aggregation of multiple Virtual Organisations [3], and therefore they do not naturally support a central point of control that could be used as a repository for resource descriptors.

In practice, when applications run in a large scale distributed system, opportunistic temporary associations tend to be created (e.g. as virtual organisations or even virtual private networks) to reflect the social or organisational structure of the institutions involved in such computations. Such opportunistic and ad-hoc associations are by excellence the type of associations supported by peer-to-peer computing. Consequently, we have undertaken the design of an agent-based peer-to-peer search system for resources in a large scale distributed environment. Peer-to-peer computing and agent-based systems are well suited for this task, as we now explain.

Our first main design decision was to make the system follow the peer-to-peer model of networking as opposed to the traditional client-server model. It avoids a central point of failure, which is a major weakness in large scale distributed

* This research is funded in part by EPSRC myGrid project (reference GR/R67743/01).

systems; this also means that the system cannot be shut down by removing a central node, as illustrated in the recent court decisions against the Napster music sharing service [4].

In a peer-to-peer system, search protocols need to be developed to use the network topology in the most efficient manner. It is not feasible for each peer to know of every other peer on the network; so, if every peer knows of a subset of all the peers, then it can communicate with them, and then each of those peers can then pass on the request to the set of peers that they know about, and so on. In this way, the request fans out from the originator and can reach many hosts in an exponential manner. For example, if each peer knows about five peers and the request is forwarded just four times, then the request will eventually reach up to $5 + 5^2 + 5^3 + 5^4 = 780$ peers. Obviously, there will be peers that know about the same peers, so this number may be slightly smaller.

Agent-based computing [5] offers the desired characteristics to implement a peer-to-peer system. Indeed, from a design viewpoint [6], agents are the right abstraction to represent peers, because of their autonomous and social natures [7]. In such a peer-to-peer context, their complex interactions can be the basis of a cooperative multi-agent system. Additionally, from an engineering viewpoint, agents systems [8] support ontologies able to provide extensible descriptions of resources, which is a fundamental property for building open systems.

A number of non trivial issues need to be investigated in order to build an agent-based peer-to-peer search system.

1. Nature of the search and how results get propagated;
2. Detection of a search termination, and cyclic search prevention;
3. Extensible and open descriptors for resources, and associated querying.

Our focus in this paper is on the *engineering* of an open and extensible peer-to-peer system. We describe our solutions to these issues, and present our design of the system we have implemented in SoFAR, the Southampton Framework for Agent Research [8]. First, we overview the key issues we address (Section 2), we then present the overall architecture (Section 3), which we follow by some design details (Section 4). We then talk about specific implementation aspects (Section 5).

2 Key Issues Overview

In this section, we overview the key issues related to building an agent-based peer-to-peer search system.

2.1 Types of Search

Two methods are available to return search results. Either results can be returned along the path the search request took, as in the Gnutella system [9], or they can be returned directly to the search originator. We can regard the former approach as *synchronous*: peers have to wait until their neighbour peers have returned

their results, to which they can add their own results, before propagating them back to the previous peers that issued the requests (cf. Figure 1). The latter approach is more *asynchronous*, as each peer can return the results directly to the originator as soon as they are available (cf. Figure 1).

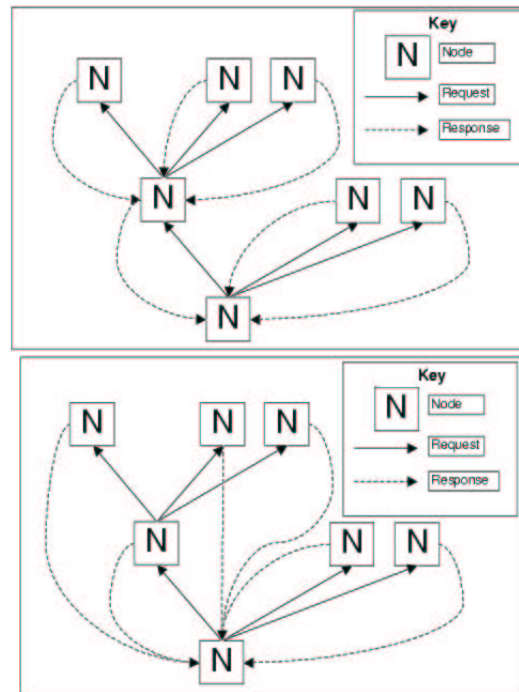


Fig. 1. Synchronous (top picture) and Asynchronous (bottom picture) Searches

With the synchronous method, all results are returned to the originator at once, which also marks the end of the search, whereas the asynchronous method returns results to the originator as soon as available, but another mechanism is necessary so that the system can indicate the end of the search.

The synchronous search provides some anonymity to the search, as peers do not know who the request originator is, nor does the originator know where the results came from. With the asynchronous search, the anonymity of the originator is lost as each peer needs to know who to send the results to.

2.2 Distributed Termination

Asynchronous search has some interesting properties: as soon as a search result is found, it can be returned to the originator. From the originator's viewpoint, results are streamed, which allows users to have an early access to information, or which allows further processing to be pipelined with the ongoing search. As

far as the functionality of a search system is concerned, it is also important to notify the user that all results have been produced, so that the user knows that there is no need to wait further. Such a facility requires an extra mechanism to indicate when all results have been delivered. This is the role of a *termination detector*, which is able to notify the originator that the following three conditions are satisfied in the whole system: (i) there is no peer left that has to process a request, (ii) there is no message in transit containing a request, (iii) there is no message in transit containing a result.

As we do not want the same peers forwarding the same request round a loop in the network, each search query will need to contain a mechanism by which each peer can decide if it has previously seen this request.

2.3 Descriptor and Query Language

Resources need to be described in such a way that they can be queried by each of the agents performing the search. To this end, we use *ontologies* to construct an “explicit specification of a conceptualisation [10]”. Ontologies are created for each aspect of a resource that needs to be described. These descriptions can then be grouped together and “attached” to resources.

We expect descriptions to be extensible and users to be able to create their own descriptions. The system should be able to support such new descriptions, as they come on line, without the need to change the search procedure, and without having to re-compile or re-deploy the system.

Describing resources is only one facet of resource discovery. There is also a need for a query language able to operate over resource descriptions. We expect this language to be simple enough to be usable by any user. A graphical user interface is desirable as it can help users who are not computer literate. The difficulty here is to define such an interface supporting the definition of queries over a description language that is extensible.

3 Overall Architecture

The agents for the system fall into two categories, those that send and receive the search requests and those that service them against the resources and their ontology descriptions.

Initially, there is an agent that sends out the created search request, marked as ‘ResultsAgent’ in Figure 2, and that is defined as the endpoint for the results being returned, as either an array of results or individual result from the peers hosting the resources. A search agent receives requests and passes them onto the peers it knows about, while at the same time performing the search on all the agents for each of the resources being shared on that peer. These are marked as ‘SearchAgent’ in Figure 2, and communicate both with other peers on the network and other agents running on the local peer.

The other agents found in the system, marked as ‘Resource SearchAgent’ in Figure 2, are designed and built to match any search query to the resources

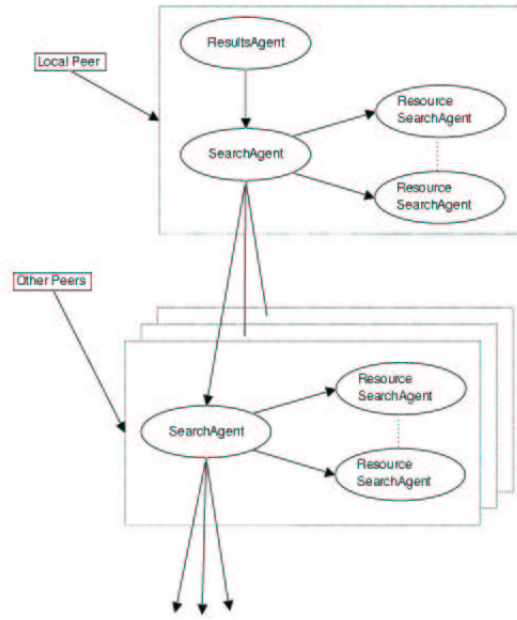


Fig. 2. Overall Architecture

they know about, whether that is a set of files on the system or other resources available to users. The results are passed back to the `SearchAgent` on the same peer, which then get passed on back to the originator in the case of the synchronous search or sent back to the originator's `ResultsAgent` in the case of the asynchronous protocol.

4 Design Details

The purpose of this section is to provide some details about our design, focusing in particular on the ontology for descriptors, the specification of queries, and the detection of distributed termination.

4.1 Descriptor Ontology

In order to be discovered, resources need to be described. To maintain some uniformity in the system, we introduce the idea of an abstract concept called a *Descriptor*. Any kind of description mechanism has to extend (in the object-oriented sense) the abstract concept `Descriptor`. These `Descriptors` can then be bundled together in order to form a resource description.

One of the resources currently available for sharing within the system is a file stored on a user's computer or file store. The descriptors currently available can describe: filenames, authors, MIME types, copyright details, and the contents of

a file. For bibliographic resources, we propose to use the Dublin Core metadata format. For Web services, WSDL descriptions of their interfaces can be used (<http://www.w3.org/TR/wsdl>).

Symmetrically, a search query can be derived from the ontology of descriptors. The SoFAR agent framework provides an ontology-based query language, which can be used by users to specify the resources they are interested in. Such a query language is based on pattern matching over ontological terms and constraints resolution. Section 5.2 describes the user interface that helps users compose queries based on such a language.

4.2 Query Specification

Both synchronous and asynchronous searches require a search query, a time-to-live argument and a unique identifier, whose roles are successively defined below.

The query is expressed in the ontology-based query language and specifies the set of descriptor patterns and constraints that resource descriptors must match. Queries are not restricted to a single pre-defined ontology; they can operate over descriptors defined at run-time provided that they express constraints over a subclass of Descriptor.

The time-to-live argument is defined as an integer and is used to control the depth of the search. A search starts off with a time-to-live value. Every time a peer forwards a query, it decrements the value of the time-to-live parameter. Once a peer receives a query with a zero time-to-live parameter, the peer no longer forwards the query. In the synchronous search, it is at this point that the results are being returned back to the search originator.

The purpose of the unique identifier is to avoid cyclic searches during which a search request would be repeatedly forwarded (within the predefined time to live) in a cycle of peers. The identifier needs to be unique across all peers in the system. For every incoming query, a SearchAgent decides if the query has been seen before; if it has not processed it before, it stores the identifier and undertakes the query.

These three parameters are all that is required for a synchronous search. For an asynchronous search, two further parameters are required, respectively to identify the query originator and a search termination detector. The first extra parameter specifies where the results should be returned to directly; this parameter typically identifies the agent that sends out the request to ensure that the results get back to the same place. The second extra parameter is used to detect the termination of an asynchronous search, which we describe in the next subsection.

The results of a search contain the set of Descriptors of each discovered resource, and the location of the resource, expressed as a URL, which can then be used to access the resource.

4.3 Distributed Termination

The problem of distributed termination detection is formulated as follows. Let us consider an asynchronous distributed system, where nodes only communicate by exchange of messages. A computation is said to be in a *terminated state* if each node has completed its local computation and if there is no message in transit. Termination detection is defined as the ability to assert that the system has reached a terminated state.

We use a well-known result [11] in distributed systems, according to which deciding whether a distributed computation is terminated can be implemented by distributed reference counting. For an asynchronous search, the originator creates a *token*, which is a newly created object that is reference counted; on the originator, a method is associated with the token, which is triggered when the token's reference counter reaches zero. When the originator initiates the query, it includes the token as part of the query; every hosts that forwards the query is also required to include the same token. When a peer terminates its search locally, it is required to "lose" its reference to the token, which in effect decrements the counter associated with it on the originator. The token's reference counter reaches the value zero when no other peer contains a reference to the token and no request with that token is in transit. Many algorithms can be used for distributed reference counting, and we are currently investigating which one would be most suitable to a peer to peer context.

An action can then be activated when the counter reaches zero, i.e., when the search has completed. An obvious application is to alert the user of the search's completion; another kind of processing on the results would be to forward results to another agent.

5 Implementation

In this Section, we describe how the system was implemented in SoFAR, the Southampton Framework for Agent Research [8]. SoFAR defines an agent as an entity implementing a set of speech acts, representing common basic communication patterns. In addition to asynchronous performatives as prescribed by FIPA and KQML agent communication languages, SoFAR also supports synchronous performatives because they are useful for querying data- and knowledge- bases or for integrating with Web Services [12].

5.1 Agents Implementation

The SearchAgent is designed to receive requests from other peers, process them locally and pass them onto the other peers it knows about. The two types of searches are supported by different performatives. A synchronous search is implemented by the performative `query_ref`, which returns the array of descriptors that match the search. Asynchronous search is supported by the performative `request`, which does not return any result.

A peer maintains a collection of proxies to agents that are running on the local system and to other peers the agent knows about. When a search query comes in, the agent starts two threads to pass the request onto the agents stored in both collections. In the synchronous search, the agent blocks until both threads have completed at which time all the results are returned back along the search chain. The asynchronous search just lets the two threads run since once they are running the main agent has no further part to play in the execution. The SearchAgent also allows other resource agents to be added to or removed from the system dynamically, through a mechanism of registration.

The FileSearchAgent is an example of an agent managing file resources. The FileSearchAgent accepts incoming search requests, checks if it knows about any files that match the search query, and returns the information to the originator along the specified route. The agent holds a persistent database mapping file-names onto resource descriptions. A database entry successfully matches a query if the descriptions contained in the entry satisfy each of the query conjuncts.

The returned result is a FileNameDescriptor which contains the name of the file that matched the query and a URL from which it can be accessed, for instance using RMI, or downloaded, for instance, using HTTP or FTP. We can therefore see that the agent framework is only used for discovering resources, while more specialised protocols are used for accessing the resource itself.

Results are returned finally to a ResultsAgent which deals with them as specified by the user. It makes use of a ResultsProcessor implementing the following three methods:

```
void add(Predicate[] results)
void add(Predicate result)
void alert()
```

The add methods are called when results are returned to the ResultsAgent, while the alert method is called when the asynchronous search is completed. Currently, there are two implemented ResultsProcessors within the system. The first one prints the results to the standard output stream. The second processor, after creating a scrolling window, prints the results in the window and generates an alert window when the asynchronous search completes.

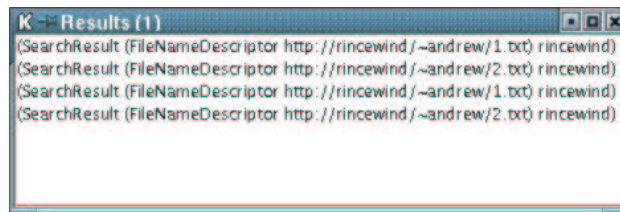


Fig. 3. Results

When results are printed in the window (cf. Figure 3), the user has the ability to select them by double clicking on them. An action can be programmed

with the selection, and can be parameterised by the type of the selected data (through a visitor pattern). When a file result is selected, the user is presented with a dialog box to choose a location where the file can be saved; then, the file is transferred from the hosting peer's computer to the user's computer.

5.2 Graphical User Interface

The main purpose of the graphical user interface is to allow a user to set up the parameters of search queries to send off into the network while waiting for the results to come in. This interface is programmed as an open system so that new descriptors can be inserted without the need to re-program or re-compile the interface.

Descriptor Creation When users have to define a new instance of a Descriptor, a window allows them to instantiate the fields of the Descriptor. The window acts as an input form whose layout is based on the definition of the Descriptor, which can be gained from the ontological definition of the Descriptor. (Reflexive methods are provided by the SoFAR framework to that effect.)

In the ontological definition of a concept, a field defined as a primitive type, e.g. string or integer, is represented as a text box in which the user can type the value they want, and which is converted into the suitable internal representation. If the field is not a primitive type, then the user is presented with a button, which when clicked on, opens a new window that contains the definition of the concept. In Figure 4, we can see the structure of a copyrightDescriptor, with a field "author" defined as a complex type (itself composed of firstName and lastName fields) and a field year defined as a primitive type.

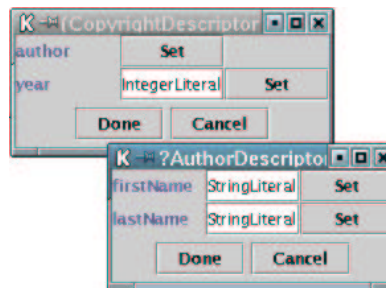


Fig. 4. Ontology setup

Search Query Creation In order to create a search query, the user needs to create a set of descriptions (or description constraints) they want in the resource. They select each of the Descriptors that is necessary from a drop down list that

is generated from the set of Descriptors known to the system. Descriptors can be added without recompiling the graphical interface. The selected Descriptor is then created and added to the descriptor set, in order to be used in the searches.

Actually, descriptors can be discovered using the search mechanism that we are describing: such a capability illustrates the reflexive nature of the system. However, a minimal set of descriptors is required (for instance, to discover descriptors), which can be found in a configuration file bundled with each peer.

Once the set of Descriptors is complete, the user selects the type of search they want (synchronous or asynchronous), and sets how far they want the search to propagate via the time-to-live parameter (cf. Figure 5). When the user clicks on the search button, the necessary search packet is constructed and passed to the ResultsAgent which fills in the rest of the necessary fields before sending the request out on the network.

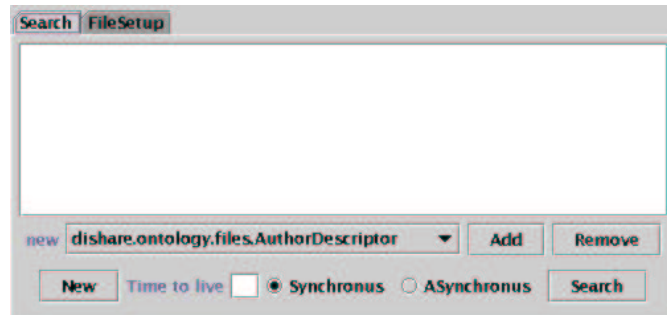


Fig. 5. Search

Publishing Window Publishing resources is made available through a user interface by which users can specify the resources to be made available for sharing. For each resource, the user is requested to attach a set of Descriptors describing the resource. The same interface is used here to edit these descriptors as when the user is creating a search request. The pairings between the files and their descriptor set are kept in a persistent database. Descriptors are allowed to be updated during program execution.

6 Related Work

Resource discovery is a central issue in distributed systems and numerous solutions exist, such as UDDI (www.uddi.org) for Web Services or Jini [13]. These solutions tend to adopt a client-server model; there are proposals to extend these so that resource information is replicated across multiple repositories.

Peer to peer systems have been widely publicised by systems such as Napster [4] and Gnutella [9]. JXTA (www.jxta.org) is a set of thin protocols for peer-to-peer computing supported by SUN Microsystems. A Google search on peer-to-peer computing also shows multiple commercial systems.

Gibbins and Hall [14] discuss the relationship between mediator-based systems for service discovery in multi-agent systems, and the technique of query routing used for resource discovery in distributed information systems. They provide a model of query routing which is used to examine the scalability of common architectures for resource discovery. The kind of search we implemented is illustrative of how we can engineer a peer-to-peer search system; our framework is suitable to implement other search algorithms as described by Gibbins and Hall.

Langley, Paolucci and Sycara [15] set out to extend the Gnutella architecture to support Agent-to-Agent communities. Dunne proposes the use of mobile agents in a manner not dissimilar to our system [16].

7 Discussion and Conclusion

Asynchronous and synchronous searches offer different properties, in terms of result availability or anonymity. In particular, the asynchronous search requires the originator to be directly contactable by peers hosting resources matching queries. This may not always be a reasonable assumption in the presence of firewalls or disconnected networks. A combination of asynchronous and synchronous searches may therefore be a suitable solution to this problem.

Throughout the paper we have discussed how the system is designed to be open and how extra ontologies or agents can be added without re-compiling or re-deploying it. This can be taken further by looking at how new descriptors can be added to a system. They can be provided as an archive that contains a listing of them to be added to the list of available descriptors for the user interface and a jar file that contains their concrete implementations. These archives can then be added to the system at run-time and used as soon as the user has got access to them on their system. These resources can be discovered, either by using the FileSearchAgent already implemented or by creating a new set of descriptors especially for the task. The results of these searches can be transferred and installed on the originator's system when selected by the user.

In this paper, we have focused on the design and engineering of a resource discovery system using an agent framework. From an engineering viewpoint, this has allowed us to provide a system that is *extensible* because new descriptions and agents can be added dynamically, and *reflexive* because the search technique can be applied to aspects of the system itself. These properties allow such a system to be long lived, without requiring it to be re-compiled or re-deployed frequently.

We are planning to experiment with such a system in two different contexts. We intend to use it to discover resources and services in the context of a Grid application (www.mygrid.org.uk) and to make information accessible in the con-

text of mobile users [17]; in particular, search results can be used as input of a recommender system.

References

1. Foster, I., Kesselman, C., eds.: *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufman Publishers (1998)
2. Cardelli, L.: Mobile computations. In: *Mobile Object Systems: Towards the Programmable Internet*. Springer-Verlag (1997) 3–6 Lecture Notes in Computer Science No. 1222.
3. Foster, I., Kesselman, C., Tuecke, S.: *The Anatomy of the Grid. Enabling Scalable Virtual Organizations*. International Journal of Supercomputer Applications (2001)
4. Napster. <http://www.napster.com> (1999)
5. Jennings, N.R., Sycara, K., Wooldridge, M.: A Roadmap of Agent Research and Development. *Int. Journal of Autonomous Agents and Multi-Agent Systems* **1** (1998) 7–38
6. Jennings, N.R., Wooldridge, M.: *Agent-Oriented Software Engineering*. In: *Handbook of Agent Technology*. AAAI/MIT Press (2001)
7. Wooldridge, M., Jennings, N.R.: *Intelligent Agents: Theory and Practice*. Knowledge Engineering Review **10** (1995)
8. Moreau, L., Gibbins, N., DeRoure, D., El-Beltagy, S., Hall, W., Hughes, G., Joyce, D., Kim, S., Michaelides, D., Millard, D., Reich, S., Tansley, R., Weal, M.: *SoFAR with DIM Agents: An Agent Framework for Distributed Information Management*. In: *The Fifth International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents*, Manchester, UK (2000) 369–388
9. Gnutella. <http://www.gnutella.com> (2000)
10. Gruber, T.R.: *Toward principles for the design of ontologies used for knowledge sharing*. Technical Report KSL-93-04, Knowledge Systems Laboratory, Stanford University (1993)
11. Tel, G., Mattern, F.: The Derivation of Distributed Termination Detection Algorithms from Garbage Collection Schemes. *ACM Transactions on Programming Languages and Systems* **15** (1993) 1–35
12. Moreau, L.: Agents for the Grid: A Comparison for Web Services (Part 1: the transport layer). In Bal, H.E., Lohr, K.P., Reinefeld, A., eds.: *Second IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2002)*, Berlin, Germany, IEEE Computer Society (2002) 220–228
13. Oaks, S., Wong, H.: *Jini In a Nutshell*. O'Reilly (2000)
14. Gibbins, N., Hall, W.: Scalability issues for query routing service discovery. In: *Proceedings of the Second Workshop on Infrastructure for Agents, MAS and Scalable MAS*. (2001)
15. Langley, B., Paolucci, M., Sycara, K.: Discovery of infrastructure in multi-agent systems. In: *Proceedings of the Second Workshop on Infrastructure for Agents, MAS and Scalable MAS*. (2001)
16. Dunne, C.R.: Using Mobile Agents for Network Resource Discovery in Peer-to-Peer Networks. *SIGecom Exchanges, Newsletter of the ACM Special Interest Group on E-Commerce*, Vol. 2.3, pages 1–9 (2001)
17. Moreau, L., Roure, D.D., Hall, W., Jennings, N.: *MAGNITUDE: Mobile AGents Negotiating for ITinerant Users in the Distributed Enterprise*. <http://www.ecs.soton.ac.uk/~lavm/magnitude/> (2001)