SoFAR: An Agent Framework for Distributed Information Management

Luc Moreau, Norliza Zaini, Don Cruickshank, David De Roure
Department of Electronics and Computer Science
University of Southampton
(L.Moreau,nmz00r,dgc,dder)@ecs.soton.ac.uk

Abstract: SoFAR, the Southampton Framework for Agent Research, is a versatile multi-agent framework designed for Distributed Information Management tasks. SoFAR embraces the notion of proactivity as the opportunistic reuse of the services provided by other agents, and provides the means to enable agents to locate suitable service providers. The contribution of SoFAR is to combine ideas from the distributed computing community with the performative-based communications used in other agent systems: communications in SoFAR are based on the startpoint/endpoint paradigm, a powerful abstraction that can be mapped onto multiple communication layers. SoFAR also adopts an XML-based declarative approach for specifying ontologies and agents, providing a clear separation with their implementation. We explain the rationale behind our design decisions; we describe *two* distributed information management applications and we recount their design and operations.

1 Introduction

The volume of information available from the World Wide Web and corporate information systems has increased dramatically over the last few years. It is now recognised that users require assistance to avoid being overwhelmed by this wealth of information (*DeRoure etal., 1996*); it is also essential that information suppliers are provided with tools that help them in authoring and maintaining it (*Carr etal., 1995*, *DeRoure etal., 1996*).

Distributed Information Management (DIM) is the term used to describe the set of activities that allow users to manage the entire life-cycle of information in a distributed environment (*Dale and DeRoure, 1997*). The activities, also referred to as *DIM tasks*, involve, amongst others, document creation and publication, information space navigation, information discovery, integrity maintenance.

The large volume of highly dynamic information involved in DIM tasks is an ideal subject for agent-style processing. This has been exemplified in several research projects, such as Pattie Maes' agents that reduce users' overload (*Maes, 1994*) or the numerous agents applied to the Internet or the WWW (*Chen and Sycara, 1998*, *Lieberman, 1995*).

Over the last decade, a series of projects at Southampton have addressed the issue of

distributed information management. This activity began with the Microcosm system (*Fountain etal., 1990*), which pioneered the idea of building a hypertext system out of a set of *loosely-coupled communicating processes*. It was an example of an open hypermedia system, in which links are regarded as first-class citizens. By managing and storing links in specific databases, called linkbases, this approach allows users to customise their information environment by selecting the appropriate linkbases. Distribution and process coordination were then investigated (*Goose etal., 1996*), and the open hypermedia philosophy was brought to the WWW by the Distributed Link Service (*Carr etal., 1995*). The same principles were also applied to other types of media, in particular to images (*Lewis etal., 1998*) and sound (*Blackburn and DeRoure, 1998*). In a project called Memoir (*DeRoure etal., 1998*), the notion of navigation trails was used to recommend documents that have been examined by users sharing similar interests. These ideas were also applied to bookmarks, annotations and document ratings shared by users (*El-Beltagy etal., 1999*). This work was further extended by using a notion of "user context" to suggest links that are relevant to users (*El-Beltagy etal., 2001*). Querying multimedia information has been an important focus in our investigation of distributed information management. We have also concentrated on optimising the actual act of query, as opposed to its content: query routing (*DeRoure etal., 1999*) has been used to optimise queries of distributed information systems, and its scalability has also been investigated (*Gibbins and Hall, 2001*). Other DIM tasks have been investigated, such as link integrity maintenance (*Moreau and Gray, 1998*) and authoring (*Carr etal., 1995*). The benefit of mobility to solve distributed information management tasks was also studied (*Dale, 1998*).

We learned two important lessons from our practical experience with designing and building prototypes over the last decade. First, it became clear that properties of weak agency identified by Wooldridge and Jennings (*Wooldridge and Jennings, 1995*), namely autonomy, social ability, reactivity and pro-activity, are also desirable for distributed information management systems. Second, we came to the conclusion that distributed information management may be regarded as the result of coordinating a multitude of simple DIM tasks. It is our belief that the functionality of the system can be the result of individual agents opportunistically exploiting services offered by other agents. Therefore, we have been working towards building a multi-agent system, where numerous agents can interact to test our hypothesis. Since individual agents would not necessarily require "intelligence" to perform their distributed information management task, we regard them as "dim" DIM agents.

In the domain of distributed information management, the ubiquitous definitions of weak agency defined in (*Wooldridge and Jennings, 1995*) are applicable, but require some qualification. We have adopted the following terminology for our DIM agent framework:

1. autonomy: the ability of an agent to effect elements of its behavioural repertoire without intervention or direct control from actors external to the agent system (e.g. the user).
2. social ability: the capacity to communicate with other agents in the system — it is an existence criterion for our framework; an agent that does not communicate, by definition, is not a participant agent.
3. pro-activity: as part of their autonomy, agents must at least possess *opportunism* as a key goal-directed behaviour; that is, they must actively search for and use the abilities of other agents to complete their tasks (*Jennings etal., 1998*).

Therefore, a multi-agent system is composed of agents with simple (usually singular) abilities who possess the above three criteria. The notion of *opportunism* enables us to build systems where agents can potentially discover new functionalities through cooperation. We believe that the simplicity of each of the DIM agents will enable the *principled engineering* of global behaviour more easily than if each agent is gifted with sophisticated functionality and behaviours — this is because the local interactions are simpler, enabling abstraction (*Jennings and Wooldridge, 2001*). By making use of other agents whenever possible, the whole is greater than the sum of the parts, so the real power of the system is realised as a result of the collective behaviour of the agents.

Over the last few years, part of our activity has concentrated on designing and building a framework for coordinating the activity of our DIM agents. The purpose of this chapter is to describe the outcome of this research, called the SoFAR framework (SOuthampton Framework for Agent Research), its properties, its design and implementation, and to present two distributed information management applications built using the framework. The framework has been used by some 60 researchers and has been the object of an undergraduate course attended by 20 students; it is also currently used in several research projects.

The key contributions of SoFAR are:

- To apply some successful ideas of the distributed computing community to multi-agent systems.
    - We adopt the same communication paradigm as Nexus (*Foster etal., 1996*), which is the communication layer that has been used to build the Computational Grid (*Foster and Kesselman, 1998*). This approach has been shown to be generic and scalable. From the agent perspective, the act of communication becomes independent of the mechanisms for communicating, which is a view that naturally extends to speech-act based communications.

    - We introduce a concept of a contract, similar to Jini leases. A contract fulfills multiple roles, including a proof of successful registration or subscription, the clearing of registries in the presence of failures, and a session identifier. Contracts are an essential element by which agents can control the flow of information that is being delivered to them.
- To adopt an XML-based declarative style for defining agents and ontologies, allowing a separation of specification from implementation.
    - XML declarations are compiled into Java classes, respectively by an “agency compiler” and an “ontology compiler”, providing a uniform and high-level programming interface. Such automatic code generation avoids the programmers to have to program repetitive code.

    - XML declarations have a clearly specified semantics, which promotes inter-operability between components. In particular, XML declarations of ontologies specify a query language relying on pattern matching and constraint resolution. Compilers can be re-targeted to other programming languages, hereby promoting open-ness in the system.

This chapter is organised as follows. In Section 2, we discuss the requirements of the framework. In Section 3, the framework itself is described, including its communication mechanism, the notion of ontology, and the architecture it provides. We then revisit the requirements and show how the framework meets them in Section 4. In Section 5, we describe an open architecture for finding information relevant to users browsing documents and a streaming application that we have implemented with this framework. Finally, we discuss related work in Section 6 and future work in Section 7, before concluding the chapter.

2 Framework for DIM Agent: Requirements

Our initial motivation is to build an advanced distributed information management system. Even though we can identify a vast number of tasks that such a system must perform, we are currently unable to define such a system precisely, nor are we able to explain its behaviour in terms of sub-components. Instead, we have adopted a bottom-up approach to building such a system. As we are able to engineer systems that perform the simple tasks that we have identified, we wish to promote the coordination of their activity, in particular by opportunistically exploiting services offered by other agents. In this Section, we present a list of requirements that we have identified for the framework in order to satisfy that goal:

1. DIM tasks need to be coordinated in a distributed environment. The number of tasks is not known a priori, and may evolve over time. The framework must be *reactive* since it must accommodate new tasks as they are created in the environment.

2. The framework must promote the opportunistic reuse of agent services by other agents. To this end, it must provide mechanisms by which agents may *advertise* their capabilities, and ways of *finding* agents supporting certain capabilities.

3. There are potentially a large number of agents that must be coordinated by the agent framework. The framework must be *lightweight* and *scalable*. By lightweight, we mean that it must be possible to implement efficient communication mechanisms, and that the administrative overhead of the framework should not hamper the overall performance of the system. By scalable, we mean that we must be able to accommodate a high number of agents (in the thousands) and that we want to avoid centralised components which would create bottlenecks during execution.

4. In order to be generic, communication primitives provided by the framework must be *independent* of the actual means of communication. Indeed, there are many communication techniques that would be suitable, such as XML messages over sockets, or object-style communications based on CORBA, DCOM or RMI. However, once an on-the-wire protocol has been chosen, it becomes very difficult to adopt another communication mechanism. Therefore, the framework is required to provide an abstract way of communicating between agents, which may be mapped onto different on-the-wire protocols.

5. Sources of information such as databases or http servers typically provide synchronous interactions, where the issuer of a query is blocked until the result is

returned. Such a type of *query-oriented communication* differs from the asynchronous type of communication usually supported by agent communication languages KQML and FIPA (*Finin etal., 1997*, *FIPA, 1999*). We want to support both mechanisms since query-oriented communications are a natural paradigm in distributed information management, whereas asynchronous communications are suitable for more loosely coupled interactions between agents.

In our framework, it is not a requirement to be directly compliant with standard agent communication languages such as KQML (*Finin etal., 1997*) or FIPA (*FIPA, 1999*). However, we believe that these standards are the result of a long experience of building agent systems, and we adopt some of their essential ideas, namely declarative communications based on speech act theory which give the context of the communication, and the organisation of knowledge into discrete ontologies.

3 The SoFAR Agent Framework: Description

In this Section, we describe SoFAR, the Southampton Framework for Agent Research. Most of the requirements of Section 2 are in fact standard distributed computing requirements, and therefore we looked at that community to find a solution to be used in the context of multi-agent systems. We present such a solution below, and we extend and adapt it to support proper agent communications, as prescribed by KQML and FIPA agent communication mechanisms, amongst others.

3.1 A Distributed Computing View

The distributed programming community has investigated numerous communication paradigms for distributed environments, such as message-passing libraries (e.g. MPI or PVM), communication channels (e.g. CSP or π-calculus), remote procedure call (RPC) and its object-oriented variant, remote method invocation (*Siegel, 1996*, *java, 1996*).

Nexus (*Foster etal., 1996*, *Moreau etal., 1997*) is a distributed programming paradigm, available as a library, which provides the essence of a distributed object system and has inspired the model of communication used in SoFAR. The communication layer Nexus has been used in the Globus projects (www.globus.org), the basis of the Computational Grid (*Foster and Kesselman, 1998*). Nexus has proven to be a generic mode of communication, which is efficient and scalable. It provides programmers with two key ideas: *startpoint/endpoint pairs* to refer to remote objects and *remote service requests* to start computations on remote objects.

In Nexus, communication flows from a communication *startpoint* to a communication *endpoint.* A startpoint is bound to an endpoint to form a *communication link*. Many startpoints can be bound to a single endpoint, in which case incoming communication is merged as in typical point-to-point message passing systems. Both startpoints and endpoints can be created dynamically; the startpoint has the additional property that it can be moved between processors using the communication operations we now describe.

A communication link supports a single communication operation: an asynchronous *remote service request* (RSR). An RSR is applied to a startpoint by providing a procedure name and some data. The RSR transfers the data to the process in which the endpoint is

located and remotely invokes the specified procedure, providing the endpoint and the data as arguments. A local address can be associated with an endpoint, in which case any startpoint associated with the endpoint can be thought of as a “global pointer” to that address.

Each communication link defines a unique communication medium, with which a specific communication method can be associated. There may be several *supported protocols*: the Nexus communication library is multi-protocol and RSRs may be transported on top of TCP, UDP and HTTP (*Michaelides etal., 1999*). In addition, each endpoint is associated with a table of handlers, from which one handler, i.e. a *method* in object-oriented terminology, is selected upon reception of an incoming RSR. In Nexus, a remote service request is a one-way communication; if results need to be returned, a second RSR has to be used.

3.2 Communications as Performatives

The Nexus programming model provides the essence of a distributed object system, with means to refer to remote objects and to activate computations on them. Jennings and Wooldridge (*Jennings and Wooldridge, 2001*) convincingly argue that agents are different to objects. We agree with their view and observe further differences as far as communications are concerned.

If we return to a message-passing view of object-oriented systems, the messages sent and received by objects typically combine the exchanged data with the intended action (a query, or perhaps a statement of change) to be performed with that data in a way that makes the two inseparable. In addition, in object-oriented systems, classes have few or no restrictions on the methods they may implement or call. By comparison, the approach taken by many agent systems is to separate intention from content in communicative acts, abstracting and classifying the former according to Searle's speech act theory (*Searle, 1969*). An agent's communications are thereby structured and constrained according to a predefined set of performatives, which together make up an *agent communication language* (ACL).

The number of different performatives varies between different ACLs. The most simple, such as Shoham's Agent-0 (*Shoham, 1993*), have less than half a dozen, while the more complex, such as KQML or FIPA have more than twenty. Our experience is that a frugal but careful choice of performatives allows our agents to interact in as complex ways as if they were using a more complex agent communication language. In particular, FIPA and KQML contain specialised performatives for tasks such as forwarding messages or issuing calls for proposals, which we respectively see as functions of the communication layer or as terms to be defined in an application ontology. At the other extreme, Agent-0 relies on the composition of basic acts to perform more complex messages, which FIPA and KQML consider as primitive. Our minimal set of performatives and their intuitive descriptions are given in Figure 1, and are an attempt to strike a compromise between these extremes, being chosen in order to avoid the complexity and communication cost that composition would entail in the most common scenarios.

`Query_if`	Does the recipient know facts which match the query?
`Query_ref`	What facts does the recipient know which match the query?
`inform`	The sender tells the recipient that the content is true
`uninform`	The sender tells the recipient that the content is false
`subscribe`	The sender asks to be informed of changes to facts which match the query
`unsubscribe`	The sender cancels a subscription
`request`	The sender asks the recipient to perform an action
`register`	The sender advertises their capabilities with a broker
`unregister`	The sender withdraws an advertisement

Figure 1: Supported Performatives

Although there are important differences between agents and objects, there are some fundamental similarities, namely that both are communicative entities. If the predominant object-oriented paradigm has shifted from message-passing to method invocation, we can similarly adopt a Nexus-like approach to inter-agent communications. In this, the performatives in Figure 1 become the names of the procedures invoked by an RSR. In addition, methods may return values to their caller in order to satisfy our requirements of query-oriented communications: simple query performatives such as `query_if` or `query_ref` return values directly, rather than through an extended message exchange involving an `inform` message sent back to the querent.

We have defined three query performatives in our ACL, each with different semantics and expected responses: `query_ref` is an exhaustive search of an agent's knowledge base which returns all terms which satisfy the query; `query_if` verifies that a statement holds; `subscribe` is a temporal query in which the querent requests that the receiver enters into a contract to inform the querent each time a statement satisfying the query changes value (e.g. is asserted or retracted).

3.3 An Agent View of Communications

The separation of intention from content is not the only difference we observe between object- and agent-based systems. Jennings and Wooldridge (*Jennings and Wooldridge, 2001*) also note that while objects control their state, they do not control their behaviour. A method invocation is an irresistible request that objects must perform. Agents do not have this compulsion, and are able to discriminate between messages based on their beliefs and the context of the message.

This *communication context* includes information about the act of communication itself such as the sender, receiver, sent time, message identifier and conversation thread. An agent may use this to reject a message, to discriminate between senders, or to determine which thread of conversation a message belongs to. This information is usually not available in object systems, but should definitely be made available in an agent system. Therefore, our model of agent communication is defined in terms of startpoints and endpoints, communication context and performatives.

A communication is based on a communication link defined by a startpoint and an endpoint. An *endpoint* identifies an agent's ability to receive messages using a specific communication protocol, and extracts messages from the communication link and passes

them on to the agent. An agent's endpoint is located where that agent resides.

A *startpoint* is the other end of the communication link, from which messages get sent to an endpoint. There may be several startpoints for a given agent, each acting as a representative of the agent at remote locations. A startpoint can be seen as a "proxy" for an agent.

As far as implementation is concerned, agents are regarded as *objects* that implement a predefined set of methods, corresponding to the performatives displayed in Figure 1. Communication between agents is performed by method invocation. Such a method is invoked on a startpoint, which takes care of packaging the method call up as a message and transmitting it to the endpoint.

Startpoints and endpoints have a crucial role: startpoints define the different components of the communication context, such as time or sender; endpoints construct the communication context and make it available to the agent. An agent is defined as an object that implements a method for each performative. Such methods are binary: the first argument is the term that is the subject of the message, whereas the second argument is the whole message itself, with its complete communication context.

Performatives such as queries are intended to return a result. The result is transmitted back to the sender agent using the communication link that carried the query, and returned as a result of the method invocation on the startpoint.

Usually, a startpoint is attached to a single endpoint, and communication is point-to-point. If a startpoint is attached to several endpoints, a multicast mode of communication becomes possible. Note that performatives that are used in multicast mode are not supposed to return a result. Such a mode of communication is particularly useful for the performative `inform` in order to propagate information to several agents using a single communication act. (The implementation may use underlying multicast primitives, or simulate multicast.)

3.4 Ontologies

The messages exchanged by agents are used to communicate information about their environment or some problem domain, and so the content of the messages must be a representation of their world. It is unreasonable to expect that all problem domains can be consistently represented by a single model, and so the design of specialised *ontologies* that form computational models of particular domains is now commonplace (*Guarino, 1998*).

Like "agent", the word "ontology" has of late become popular with the computing community, and its meaning has become more vague as its use has increased. Gruber stated one of the best known definitions "*An ontology is an explicit specification of a conceptualisation*" (*Gruber, 1993*). He goes on: "...we can describe the ontology of a program by defining a set of representational terms. In such an ontology, definitions associate the names of entities in the universe of discourse (e.g. classes, relations, functions, or other objects) with human-readable text describing what the names mean, and formal axioms that constrain the interpretation and well-formed use of these terms. Formally, an ontology is the statement of a logical theory." (*Gruber, 1993*) Guarino and

Giaretta have criticised the notion of conceptualisation used by Gruber (*Guarino and Giaretta, 1995*); they define a (formal) ontology as the set of formulas that are considered to be always true (and therefore sharable among multiple agents), independently of particular states of affairs.

Both definitions highlight an essential property of ontologies: they are a *shared understanding* of some domain that can be communicated across people and computers. A practical consequence of this property is that ontologies can be shared and reused among different applications (*Farquhar etal., 1996*); in particular, we believe that they are attractive in agent-based distributed information management.

Pragmatically, an ontology is constituted by a specific vocabulary used to describe a certain reality, plus a set of explicit assumptions regarding the intended meaning of the vocabulary words. Such a vocabulary may of course be used to compose assertions to be exchanged between agents, but also to express queries or requests related to the domain.

In SoFAR, ontologies are organised along a hierarchy based on single inheritance. Terms of ontologies are defined by the unique parent they extend and a (possibly empty) set of typed fields they contain. Terms are defined using an XML syntax. For instance, a `Person` can be defined as an entity composed of three fields.

```
<term name="Person" extends="Entity">
  <field type="String" name="title"/>
  <field type="String" name="personal"/>
  <field type="String" name="family"/>
</term>
```

It is sometimes convenient to define a term as abstract, which essentially declares a type, for which there cannot be any instance. An example of an abstract term is `Entity`, which is extended by two terms `Person` and `Group`.

```
<term name="Entity" extends="Predicate" abstract="yes">
</term>

<term name="Group" extends="Entity">
  <field type="String" name="name"/>
</term>
```

The root of the hierarchy is the type `Term`, which is also abstract: any concept or relation in a SoFAR ontology is an extension of `Term`. Additionally, we introduce a notion of `Predicate` that is a kind of `Term` we can query about. SoFAR supports the usual primitive types found in most programming languages, such as integers, floats, booleans, and strings.

3.5 Ontology-Based Query Language

The ontological definitions allow us to define typed data structures, but also they provide the foundation of a query language over sets of such data structures. The benefits of this approach is a uniform handling of ontological terms and queries over them. Our query language is based on pattern-matching and requires adding *variables* and *constraints* to ontological definitions.

Let us consider an instance of the `Person` term defined in the previous section.

Person("Dr","Luc","Moreau")

If we regard this term as a query to an agent *A*, it has the following meaning: is a `Person` with title `Dr`, first name `"Luc"` and family name `Moreau` known to the agent *A*?

Any of the fields can be replaced by a typed variable. For instance, the following term denotes all the persons with a first name `"Luc"`.

Person(?*String*,"Luc",?*String*)

Variables are not restricted to primitive types, but can be used to denote any terms of an ontology. The following query is expected to return all the persons with a first name `"Luc"` and their associated group.

InGroup(*Person*(?*String*,"Luc",?String),?Group)

In order to make the language more expressive, we can attach constraints to variables. For instance,

Person(?*String*,λ *x*:?*String*. *x*≠"*Luc*",?*String*)

denotes the set of `Person`s with a first name that is not `"Luc"`. Note that our use of a λ-expression is purely for notational convenience: it simply means that the argument is expected to be of type `String` and must differ from the string `"Luc"`.

Any type of constraint can be programmed by users, who just need to implement a method that determines whether the constraint is satisfied. In fact, all variables are associated with a constraint: the `Universal` constraint is the least constraining of all constraints because it is satisfied for any term. Additionally, logical combinators `not`, `and`, `or` are provided as constraints.

We have formally defined a pattern-matching algorithm over the ontology-based language we have presented in this section. Several variants were defined, depending on whether inheritance is supported in the algorithm.

3.6 Contractual Registration and Subscription

Registration is the action by which an agent declares to the *registry* agent its ability to handle some messages. If the registry answers positively to a registration act, it commits itself to advertise the registered capability and to return it to agents which ask matching queries. As a proof of its commitment, the registry issues a *contract* as a result of the registration act. As long as the contract remains live, the registry will retain the advertised capability. Conversely, if the agent that registered the capability desires to stop its advertising, it just has to terminate the associated contract.

A similar mechanism exists for subscriptions. If an agent decides to answer positively to a subscription act, it commits itself to honour such a subscription: whenever a fact changes it informs the interested subscribee. For each successful subscription act, a contract is issued as a proof of commitment. The subscribee just needs to terminate the contract in order to suspend the flow of `inform` messages.

The goal of the agent framework is to promote agent reuse by information sharing between agents. In an environment composed of numerous agents, there must be some means of avoiding being swamped by irrelevant information; two different ways are provided by the framework. *(i)* The general algorithm for matching and constraints

satisfaction allows agents to declare interests that are very specific, and to be informed of facts satisfying them. *(ii)* Contracts allow agents to terminate a flow of information when suitable.

There are several ways by which an agent can find information. They differ by when the result is returned, and by the agent's ability to control the flow of information. *(i)* Exhaustive searches (performative `query_ref`) and specific queries (performative `query_if`) complete their execution with the requested information. *(ii)* An agent *A* can advertise (performative `register`) its desire to be informed about a given topic. Any agent in the system may inform *A* on the topic. Agent *A* is given little control over the flow of information. It can certainly stop advertising its interest, but there is no requirement for the other agents to stop propagating information to *A*. *(iii)* In order to gain more control of the flow of information, agent *A* can subscribe (performative `subscribe`) to those agents who are knowledgeable on the topic. In return, each of these agents issues a contract, which may be used to terminate the individual subscriptions.

3.7 XML Agent Specification

The framework as it stands is powerful enough to support any form of interaction in a multi-agent system. However, the programming interface is still rather low-level: the programmer is required to repeat identical code too often, which makes the programming tedious. For instance, many agents need to find out the Registry and to advertise their capabilities; many agents need to find about other agents by interrogating the Registry and to query these agents; many agents implement some of the performatives of Figure 1, and need to check whether the arguments received have the required type, and so on.

In order to facilitate the programming, the maintenance, the building and dissemination of agents, SoFAR offers an "agency compiler", which takes an XML specification of a set of agents, and provides: *(i)* A Java template, ready to be subclassed, implementing many of the tedious operations that all agents have to implement; *(ii)* an agent definition ready to be stored in a database for future reference; *(iii)* files identifying the permissions to be granted to agents; *(iv)* a Makefile able to compile the agents, bundle them in a jar file, and export them for download.

```
<agent name="Index" extends="NullAgent">
<author>Liza</author>
<package>sofar.magnitude.index</package>

<comment>
An agent able to get summarize url and parse according to keywords.
</comment>

<import>sofar.ontology.base.*</import>
<import>sofar.magnitude.ontology.*</import>

<communication>rmi</communication>
<ontology>sofar.magnitude.ontology.Magnitude</ontology>

<dispatch>
  <method name="inform">
    <type>LinkRequestInfo</type>
  </method>
</dispatch>
</agent>
```

As an example, Figure 2 shows the declaration of an agent support RMI communication, and accepting the inform performative, with predicate LinkRequestInfo. Experience has shown that such XML declarations offer a clear separation between agent specification and implementation, which facilitates the development of agents.

4 Requirements Revisited

In this Section, we examine how the design of the SoFAR framework satisfies the requirements we enumerated in Section 2.

1. At the framework level, reactivity of the system can be implemented by the subscription mechanism, by which agents ask to be informed about facts, when they change. From an architectural viewpoint, as agents advertise or retract services, this information will be propagated between registries, and passed to agents that have subscribed to this type of information.

2. The registry and the subscription mechanisms allow agents to advertise their capabilities, in order to be reused by other agents. Agents use the registry to take opportunistic advantage of agents running in the system. Contracts allow agents to exercise control on the flow of information that is directed to them.

3. Even though the communication mechanism abstracts away from the communication details, the framework remains lightweight. The cost of this abstraction is two additional method invocations (one at the startpoint and one at the endpoint), which can be neglected compared to the cost of communication. Furthermore, by adopting a predefined set of performatives and typed ontologies, we reduce interpretation of messages, which makes their processing lightweight. In order to make the framework scalable, we have avoided centralised routing of messages: communications are point to point. A multicast mode of communication can even be implemented (though, currently, multicasting is simulated). We use replication of data in order to distribute the content of the registry. Other techniques such as query routing (*DeRoure etal., 1999*) or hierarchical organisation (*Moreau, 1998*) are being investigated.

4. Currently, our implementation relies on shared memory communications, Java rmi (*java, 1996*), and SOAP (*Gudgin etal., 2001*). We are investigating other communication technologies such as IPv6 and CORBA.

5. The performatives `query_ref` and `query_if` provide the query-oriented type of communications suitable for distributed information management. The subscription mechanism and associated `inform` messages offer an asynchronous alternative, in the spirit of KQML and FIPA ACLs.

5 Agent-Based Distributed Information Management

In this Section, we present two distributed information management applications that we have developed using SoFAR. Others have been developed such as (*Weal etal., 2001*), and the framework is currently used in several research projects.

5.1 Magnitude

The aim of the Magnitude project (Mobile AGents Negotiating for ITinerant Users in the Distributed Enterprise) is to investigate the use of mobile agents in the design of the "pervasive information fabric" (*Thompson etal., 2000*). In a first step, we have designed and built a distributed information management system providing related information to users, as they access information with their browser; initially, the mobility aspect is not addressed in this architecture. We use agent technology with the purpose of making the system open so that many agents can be added dynamically to the system.

At present, the behavior of the information system is visible to users in the browser sidebar. While users navigate information in the main browser window, the sidebar displays links to documents that are relevant to the document currently displayed in the main window; we call such links *related links*. The system is multi-user, and allows a community of users to share information.

The information system architecture is displayed in Figure 3; it can be divided into three main parts: (A) Agents responsible for managing information; (B) Agents responsible for extracting information; (C) Community memory agents.

Information management The system offers users the possibility to bookmark documents, and references these, which we call *links*, are stored into the community memory. Two types of links are distinguished. *Static links* refer to pages whose content does change, whereas *live links* are associated with pages whose content is regularly updated by the owner site. For example, a link to a news site such as www.bbc.co.uk is regarded as live. We define an *Information Manager* Agent as an agent able to manage the information associated with a link. Currently, two instances are defined. The *Document Manager Agent* is an agent responsible for handling static links, whereas the *News Reporter Agent* is able to refresh the information associated with live links.

All Information Manager Agents are required to subscribe to the Browser Agent in order to be informed of the specific information they are interested in. For example, the Document Manager Agent subscription allows it to receive information on requests to add or remove static links from the community memory. The subscription method allows more information manager agents to take part in the system to handle other types of links, such as for example an Image Manager Agent that handles image links.

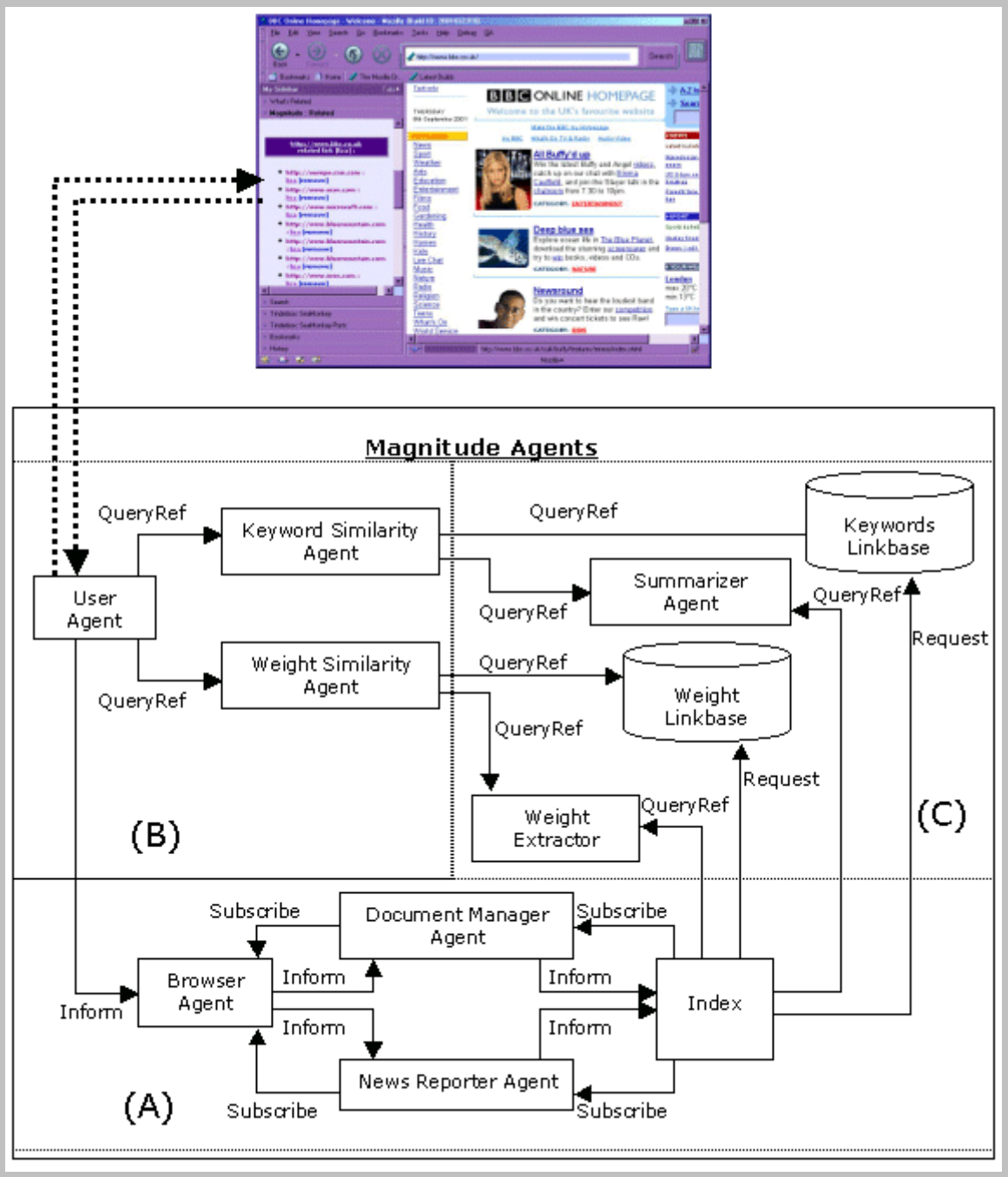

Figure 3 – Magnitude declaration

Information Extraction There is not a single method to determine whether two documents are related. Therefore, we introduce the idea of an abstract relationship "*SimilarLinks*" that represents the fact that two links are related. A single query of the form *SimilarLinks*("currentDOC",?URL) is issued in order to find out all the links that are related to the current document. Agents able to handle such a request are called *Similarity Agents*. Currently, we have defined two similarity agents, which determine similarity according to keywords and weight vectors, respectively. The *Keyword Similarity Agen*t returns urls to documents sharing similar keywords, while the *Weight Similarity Agent* applies a method introduced in (*El-Beltagy etal., 2001*). For the Weight Similarity Agent, the similarity of two documents is determined by using the cosine similarity function, over a vector representation computed by the "term frequency,

inverse document frequency" method. Two documents are stated to be similar if the value of similarity is more than a threshold value. There are many other ways that one document can be similar to another, for example the similarity measure could be based on the document's context or on a colour histogram.

The *User Agent* is the agent in charge of constructing queries based on the requests from the browser sidebar to all agents advertising the ability to handle queries on the abstract relationship "SimilarLinks". Thus, if a new Similarity Agent supporting a different type of similarity were created, it would be straightforward to plug it into the system, as the User Agent would at once be aware of it.

Community Memory The community memory consists of Linkbases and Information extractor agents. These agents serve the need of Similarity Agents by providing the required information and carry out requests from management agents such as to add or remove links from the Linkbases. Linkbases act as repositories for all links bookmarked by users in the community. At creation time, the user decides whether links are public or private; public links are shared by all users and can be created anonymously or with the user's name. Private links are visible only by their creator.

Conclusion As illustrated above, it is easy to add new agents to the system. Such an extensible system gives users greater chances to get higher quality of related links. However, a possibility that we cannot overlook is that the User Agent may be swamped by too much information from all the Similarity Agents. Thus, we are currently investigating a market economy as a means of selecting only the highest quality of information by the most efficient Similarity Agents.

5.2 HyStream

HyStream addresses distributed information management in the context of live information flows including multimedia. The purpose of the HyStream project is to augment streamed media with metadata carrying information about data items. Such metadata is itself carried in a separate stream, and we refer to it as *temporal metadata*. For example, a live news broadcast can be augmented with temporal metadata carrying information about news items. The metadata is temporal because it refers to a particular time during the broadcast. The temporal metadata could also include catalogue information for news footage, information about rights to use the material, subtitles and links to associated resources such as online documents. Agents can use the metadata to filter new items or bring them to the user's attention, translate subtitles and perform appropriate searches and customisation.

An architectural diagram of the HyStream system is shown in Figure 4. The agents that serve synchronised metadata are labelled *service agent*, and the agent that connects to the user's web browser is labelled *user agent*. The service agent is synchronised with the user agent using a specifically-designed synchronisation protocol (*Cruickshank etal., 2001*), so that each sender can determine when a piece of metadata must be sent in order to reach the user agent in time for the corresponding event in the media.

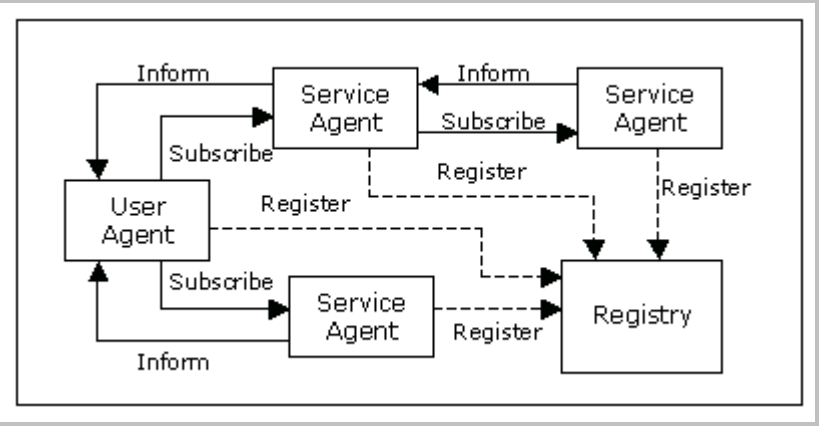

Figure 4 - Architectural Diagram of HyStream agents

In terms of performatives, the user agent initiates a "session" by *subscribing* to a service agent that has registered its capability to serve the HyStream protocol. If the subscription is successful, then a contract between the user agent and the service agent is made. The service agent will then send temporal metadata to the receiving agent when they become applicable to the media stream. The contract is a first-class object that is returned to the user agent as a consequence of a successful subscription. When the subscription is no longer required, the user agent terminates the contract, which causes SoFAR to notify the service agent that the subscription has terminated. The contract is also used as a session identifier in the synchronisation protocol between the user agent and the service agent.

For the transmission of temporal metadata, we opted for *just in time* delivery. The server sends data to the user just in time for it to be used. Delivering metadata just in time means that the system as a whole is reactive. In our news channel scenario, the process of authoring metadata and the delivery of the metadata to the user agent are simultaneous processes. By leaving the transmission of metadata until the last moment, just before it is applicable to the media stream, ensures that the metadata is up-to-date. A service agent can determine the latest time that it can send a piece of metadata to be utilised by the user agent. If that time has already passed, then the metadata is *late*. In this case, the metadata is not useful to the user agent and is simply dropped.

In a wide area network, we need mechanisms to control the scope of service discovery and advertisement of capabilities. We consider that the framework might extend across the Internet, comprising of thousands of nodes separated by varying network latencies. We cannot expect a user agent to search every node for potential servers, nor can we expect a service agent to advertise its service on every node. A scalable solution is achieved by the service agent choosing the number of nodes on which to advertise its service.

Conclusion In the HyStream project, the SoFAR framework has a number of benefits. The distributed and concurrent nature of SoFAR agents allows us to place agents close to video sources and capture metadata, and to communicate that metadata through the framework to the end consumer. The contracts that represent the commitment to each metadata subscription allow us a handle on each subscription, and provide useful session identifiers between agents. The pipeline architecture of HyStream has allowed us to implement agents that inject new metadata from other sources, e.g. the Magnitude agents, into existing streams.

6 Discussion and Related Work

The exact nature and requirements of agency are still contentious subjects, with some disagreement in the literature. We follow Jennings and Wooldridge (*Jennings and Wooldridge, 2001*) for our view of agency, regarding it as a software engineering tool for managing the complexity of system development. Nwana and Ndumu (*Nwana and Ndumu, 1999*) raise several points, namely that the standardised ACLs contain too many performatives, some of which are used only infrequently, and that the effects on ontology design of the interactions between a problem domain and a task to be accomplished are underinvestigated. If, as they suggest, the short term solution is to create only limited domain ontologies, we believe that our use of mixed ontology expressions is a useful approach to bridging the gap between limited ontologies and broader general-purpose ontologies.

SoFAR is not the only Java-based agent framework; there exist a number of others, the most notable of which are Zeus (*Nwana etal., 1999*), JAFMAS (*Chauhan, 1997*), JATlite (*Petrie, 1996*), fipa-os (*fipaos, 1999*), Ajanta (*Tripathi etal., 1999*) and JACK (Jack, 1999). Zeus and JAFMAS adopt a similar approach, providing both a FIPA- or KQML-based communications infrastructure and a planning engine for handling rule-based conversations by means of automata models, and as such are representative of a "traditional AI" approach to agency. JATlite also provides KQML-based messaging, but is flexible: it is designed to support other ACLs as necessary and does not place any restrictions on the internals of the agents. fipa-os (*fipaos, 1999*) is a FIPA-compliant platform, which necessarily relies on a CORBA-based communication substrate; our approach can use CORBA as well as other technologies. Ajanta uses a method invocation approach not unlike ours, but does not constrain the methods used in performatives. JACK is a Java-based BDI framework, which provides facilities for formulating plans and reasoning about goals, but does not consider the pragmatics of communication or distribution issues.

In its parsimonious approach to its ACL and the simplicity of its agents, SoFAR is most like Agent-0 (*Shoham, 1993*) and the systems derived from it, such as AgentBuilder (*AgentBuilder, 1999*) or PLACA (*Thomas, 1994*), although SoFAR does not provide support for planning abilities at a framework/language level as this latter system does. AgentBuilder is noteworthy as a commercial framework based on Shoham's notion of *agent-oriented programming* (*Shoham, 1993*), but using KQML as its ACL rather than the much simpler Agent-0.

7 Future Trends

We are now observing a convergence of different technologies. Web services are a distributed architecture under consideration by the World Wide Web consortium, based on XML protocols (*Gudgin etal., 2001*), and are particularly targeted at e-Business applications.

The Grid paradigm is meant to enable the access to computing power and resources with the ease similar to electrical power (*Foster and Kesselman, 1998*). The Computational

Grid is analogous to the electric power Grid, because it offers a consistent access to geographically distributed resources, irrespective of their physical location or access point. A number of Grid services have already been implemented as Web services (*Foster and Kesselman, 1998*), bridging the gap between these two technologies.

Grid technologies are enabling e-Science, the type of science that is collaborative and multi-disciplinary involving teams spanning institutions, states, countries and continents. The "Grid problem" is defined as flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions and resources, which is also referred to as *virtual organisations* (*Foster etal., 2001*). Multi-Agent systems are particular suited for the design of flexible virtual organisations (*Jennings etal., 2000*), though *scalable* protocols for forming, running and disbanding virtual organisations are still needed in the context of the Grid. With a communication layer based on SOAP, SoFAR is an opening of the agent world to Web services.

In order to provide higher-level multi-agent protocols, an agent framework should provide mechanisms that allow their definition and experimentation in an easy manner. Using our XML approach for declarative definition, we have prototyped a *protocol compiler* that transforms an XML definition of a protocol into a transition table directly interpretable by an agent.

The deployment of multi-agent systems in the context of the Grid, but also in the context of information systems we study in the Magnitude and HyStream projects requires an infrastructure able to cope with wide-area networks. In particular, the issue of scalability in directory services, both for looking up and advertising information becomes critical. New modes of communication may also have to be investigated. Multicasting will allow us to propagate streams of `inform` messages in a scalable way. Agents will also be allowed to negotiate the quality of service (QoS) they require to communicate the data they manipulate. Finally, security techniques such as encryption and authentication will be integrated in the communication model.

The inexorable trend towards the pervasive computing environment emphasises the need for mobile user support; in particular, a piece of technology that seems promising is the concept of mobile agent, acting as a mobile representative of a mobile user. Security and communication issues regarding mobile agents need to be addressed, before they can be deployed in a e-Business environment.

8 Conclusion

We have designed and implemented an agent communication mechanism that is derived from distributed computing techniques, but taking on board the reality of agency. Our approach is general and abstracts away from the communication details, supporting several on-the-wire protocols; it is lightweight and a proven route to scalability. In order to promote opportunistic reuse of agent services by other agents, our framework provides mechanisms to advertise information, query agents and automatically manage subscriptions. A set of ontologies has been defined in order to support distributed information management tasks.

The SoFAR framework has been the focus of a tremendous activity involving up to sixty researchers in the Intelligence, Agents Multimedia Group at Southampton. Training

sessions were organised about agents, ontologies, and the actual framework implementation in Java. On three occasions, a group activity, called "agentfest", took place: during a three day session, those researchers developed agents. As a result, SoFAR has now been adopted by several researchers for their everyday research. The framework has also been used for undergraduate teaching. We welcome other researchers to try SoFAR, which is available from `www.sofar.ecs.soton.ac.uk`.

9 Acknowledgements

This research is supported in part by EPSRC and QinetiQ project "Magnitude" reference GR/N35816, EPSRC project "HyStream" reference GR/M8407/01 and EU Project IST-2000-26135 FEEL. Some 60 researchers and 20 undergraduate students have used and developed SoFAR agents and we wish to acknowledge their contribution. We also wish to thank Nick Gibbins for his initial contribution to the design of SoFAR.

References

(AgentBuilder, 1999)

AgentBuilder: an Integrated Tookit for Constructing Intelligent Software Agents. Reticular Systems, Inc. Available from http://www.agentbuilder.com/. 1999.

(Blackburn and DeRoure, 1998)

Blackburn, S. G. and DeRoure, D. C. (1998). A tool for content based navigation of music. In *Proceedings of ACM Multimedia '98*, pages 361--368. ISBN: 1-58113-036-8.

(Carr etal., 1995)

Carr, L. A., DeRoure, D. C., Hall, W., and Hill, G. J. (1995). The distributed link service: A tool for publishers, authors and readers. In *Fourth International World Wide Web Conference: The Web Revolution, (Boston, Massachusetts, USA).*, pages 647--656. O'Reilly & Associates. Appears in World Wide Web Journal issue 1, ISBN 1-56592-169-0, ISSN 1085-2301.

(Chauhan, 1997)

Chauhan, D. (1997). *JAFMAS: A Java-based Agent Framework for Multiagent Systems Development and Implementation*. PhD thesis, ECECS Department, University of Cincinnati.

(Chen and Sycara, 1998)

Chen, L. and Sycara, K. (1998). WebMate: a Personal Agent for Browsing and Searching. In *Proceedings of the Second International Conference on Autonomous Agents*, pages 132--139.

(Cruickshank etal., 2001)

Cruickshank, D., Moreau, L., and Roure, D. D. (2001). Architectural Design of a Multi-Agent System for Handling Metadata Streams. In *The fifth ACM International Conference on Autonomous Agents*, Montreal, Canada.

(Dale, 1998)
Dale, J. (1998). *A Mobile Agent Architecture for Distributed Information Management*. PhD thesis, University of Southampton.

(Dale and DeRoure, 1997)
Dale, J. and DeRoure, D. (1997). Towards a Framework for Developing Mobile Agents for Managing Distributed Information Resources. Technical Report M97/1, University of Southampton.

(DeRoure etal., 1996)
DeRoure, D., Hall, W., Davis, H., and Dale, J. (1996). Agents for distributed multimedia information management. In *Pratical Application of Intelligent Agents and Multi-Agent Systems*, pages 91--102, London, UK.

(DeRoure etal., 1998)
DeRoure, D., Hall, W., Reich, S., Pikrakis, A., Hill, G., and Stairmand, M. (1998). An open framework for collaborative distributed information management. In *Seventh International World Wide Web Conference (WWW7), Brisbane, Australia*, volume 30, pages 624--625. Elsevier. Published in Computer Networks and ISDN Systems.

(DeRoure etal., 1999)
DeRoure, D. C., El-Beltagy, S., Gibbins, N. M., Carr, L. A., and Hall, W. (1999). Integrating link resolution services using query routing. In *5th Workshop on Open Hypermedia Systems (OHS5)*, Darmstadt, Germany.

(El-Beltagy etal., 1999)
El-Beltagy, S., DeRoure, D. C., and Hall, W. (1999). A multiagent system for navigation assistance and information finding. In *THe Fourth International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, pages 281--295.

(El-Beltagy etal., 2001)
El-Beltagy, S., Hall, W., Roure, D. D., and Carr, L. (2001). Linking in context. In *Proc The Twelfth ACM Conference on Hypertext and Hypermedia (Hypertext '01)*, pages 151--160. ACM, ACM Press.

(Farquhar etal., 1996)
Farquhar, A., Fikes, R., and Rice, J. (1996). The ontolingua server: A tool for collaborative ontology construction.

(Finin etal., 1997)
Finin, T., Labrou, Y., and Mayfield, J. (1997). *Software Agents, J. Bradshaw, Ed.*, chapter KQML as an Agent Communication Language. MIT Press.

(FIPA, 1999)
FIPA. FIPA: Foundation for Intelligent Physical Agents. `http://www.fipa.org`.

(fipaos, 1999)
fipaos (1999). FIPA-OS.
`http://www.nortelnetworks.com/products/announcements/fipa`.

(Foster and Kesselman, 1998)
Foster, I. and Kesselman, C., editors (1998). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufman Publishers.

(Foster etal., 1996)
Foster, I., Kesselman, C., and Tuecke, S. (1996). The Nexus Approach to Integrating Multithreading and Communication. *Journal of Parallel and Distributed Computing*, 37:70--82.

(Foster etal., 2001)
Foster, I., Kesselman, C., and Tuecke, S. (2001). The anatomy of the grid. enabling scalable virtual organizations. *International Jounral of Supercomputer Applications*.

(Fountain etal., 1990)
Fountain, A. M., Hall, W., Heath, I., and Davis, H. C. (1990). MICROCOSM: An Open Model for Hypermedia With Dynamic Linking. In Rizk, A., Streitz, N., and André, J., editors, *Hypertext: Concepts, Systems and Applications (Proceedings of ECHT'90)*, pages 298--311. Cambridge University Press.

(Gibbins and Hall, 2001)
Gibbins, N. and Hall, W. (2001). Scalability issues for query routing service discovery. In *Proceedings of the Second Workshop on Infrastructure for Agents, MAS and Scalable MAS*.

(Goose etal., 1996)
Goose, S., Dale, J., Hill, G. J., DeRoure, D. C., and Hall, W. (1996). An Open Framework for Integrating Widely Distributed Hypermedia Resources. In *Third IEEE Conference on Multimedia Computing and Systems (ICMCS'96)*, Hiroshima, Japan,.

(Gruber, 1993)
Gruber, T. R. (1993). Toward principles for the design of ontologies used for knowledge sharing. Technical Report KSL-93-04, Knowledge Systems Laboratory, Stanford University.

(Guarino, 1998)
Guarino, N. (1998). Formal ontology and information systems. In *Formal Ontology in Information Systems: Proceedings of FOIS'98*. IOS Press.

(Guarino and Giaretta, 1995)
Guarino, N. and Giaretta, P. (1995). Ontologies and knowledge bases: Towards a terminological clarification. In Mars, N., editor, *Towards Very Large Knowledge Bases*. IOS Press.

(Gudgin etal., 2001)
Gudgin, M., Hadley, M., Moreau, J.-J., and Henrik Frystyk Nielsen, editors (2001). Soap version 1.2. Technical report, World Wide Web Consortium.

(Jack, 1999)
JACK Intelligent Agents User Guide. Agent Oriented Software Pty. Ltd., 1999.

(Java, 1996)
Java Remote Method Invocation Specification. Sun Microsystems. 1996.

(Jennings etal., 2000)
Jennings, N. R., Faratin, P., Norman, T. J., O'Brien, P., Odgers, B., and Alty, J. L. (2000). Implementing a business process management system using adept: A real-world case study. *Int. Journal of Applied Artificial Intelligence*, 14.

(Jennings etal., 1998)
Jennings, N. R., Sycara, K., and Wooldridge, M. (1998). A roadmap of agent research and development. *Int Journal of Autonomous Agents and Multi-Agent Systems*, 1(1):7--38.

(Jennings and Wooldridge, 2001)
Jennings, N. R. and Wooldridge, M. (2001). *Handbook of Agent Technology*, chapter Agent-Oriented Software Engineering. AAAI/MIT Press.

(Lewis etal., 1998)
Lewis, P. H., Kuan, J., Perry, S., Dobie, M. R., Davis, H. C., and Hall, W. (1998). Content based navigation from images. *Journal of Electronic Imaging*, 7(2):275--281.

(Lieberman, 1995)
Lieberman, H. (1995). Letizia: An agent that assists web browsing. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Montreal, Canada.

(Maes, 1994)
Maes, P. (1994). Agents that Reduce Work and Information Overload. *Communications of the ACM*, 37(7):31--40.

(Michaelides etal., 1999)
Michaelides, D., Moreau, L., and DeRoure, D. (1999). A Uniform Approach to Programming the World Wide Web. *Computer Systems Science and Engineering*, 14(2):69--91.

(Moreau, 1998)
Moreau, L. (1998). Hierarchical Distributed Reference Counting. In *Proceedings of the First ACM SIGPLAN International Symposium on Memory Management (ISMM'98)*, pages 57--67, Vancouver, BC, Canada. Also in *ACM SIGPLAN Notices*, 34(3):57--67, March 1999.

(Moreau, 1999)
Moreau, L. (1999). Distributed Directory Service and Message Router for Mobile Agents. Technical Report ECSTR M99/3, University of Southampton.

(Moreau etal., 1997)
Moreau, L., DeRoure, D., and Foster, I. (1997). NeXeme: a Distributed Scheme Based on Nexus. In *Third International Europar Conference (EURO-PAR'97)*, volume 1300 of *Lecture Notes in Computer Science*, pages 581--590, Passau, Germany. Springer-Verlag.

(Moreau and Gray, 1998)
Moreau, L. and Gray, N. (1998). A Community of Agents Maintaining Links in the World Wide Web (Preliminary Report). In *The Third International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents*, pages 221--235, London, UK.

(Moreau and Queinnec, 1997)
Moreau, L. and Queinnec, C. (1997). Design and Semantics of Quantum: a Language to Control Resource Consumption in Distributed Computing. In *Usenix Conference on Domain-Specific Languages (DSL'97)*, pages 183--197, Santa-Barbara, California.

(Nwana and Ndumu, 1999)
Nwana, H. and Ndumu, D. (1999). A perspective on software agents research. *The Knowledge Engineering Review*.

(Nwana etal., 1999)
Nwana, H., Ndumu, D., Lee, L., and Collis, J. (1999). Zeus: A tool-kit for building distributed multi-agent systems. *Applied Artifical Intelligence Journal*, 13(1):129--186.

(Petrie, 1996)
Petrie, C. (1996). Agent-based engineering, the web, and intelligence. *IEEE Expert*.

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

(Shoham, 1993)
Shoham, Y. (1993). Agent-oriented Programming. *Artificial Intelligence*, 60:51--92.

(Siegel, 1996)
Siegel, J. (1996). *CORBA fundamentals and programming*. Wiley.

(Thomas, 1994)
Thomas, S. R. (1994). The PLACA Agent Programming Language. In Wooldridge, M. J. and Jennings, N. R., editors, *ECAI-94 Workshop on Agent theories, architectures and languages*, volume 890 of *Lecture Notes on AI*. Springer-Verlag.

(Thompson etal., 2000)
Thompson, M., Roure, D. D., and Michaelides, D. (2000). Weaving the Pervasive Information Fabric. In Reich, S. and Anderson, K., editors, *6th International Workshop on Open Hypermedia Systems and Structural Computing (OHS-6)*, volume 1903 of *Lecture Notes in Computer Science*, pages 87--95, San Antonio, Texas, USA. Springer-Verlag.

(Tripathi etal., 1999)

Tripathi, A., Karnik, N., Vora, M., Ahmed, T., and Singh, R. D. (1999). Ajanta -- A Mobile Agent Programming System. Technical Report TR98-016, Department of Computer Science, University of Minnesota.

(Weal etal., 2001)

Weal, M., Hughes, G., Millard, D, and Moreau, L. (2001). Open Hypermedia as a Navigational Interface to Ontological Information Spaces. Proceedings of the Twelveth ACM Conference on Hypertext and Hypermedia (HT'01), Aarhus, Denmark.

(Wooldridge and Jennings, 1995)

Wooldridge, M. and Jennings, N. R. (1995). Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, 10(2).