SoFAR with DIM Agents

An Agent Framework for Distributed Information Management

Luc Moreau, Nick Gibbins, David DeRoure, Samhaa El-Beltagy,
Wendy Hall, Gareth Hughes, Dan Joyce, Sanghee Kim,
Danius Michaelides, Dave Millard, Sigi Reich, Robert Tansley, Mark Weal
Multimedia Research Group
University of Southampton
L.Moreau, nmg97r@ecs.soton.ac.uk

Abstract

In this paper we present SoFAR, 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 some 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, which is the foundation of Nexus, the communication layer at the heart of the Computational Grid. We explain the rationale behind our design decisions, and describe the predefined set of agents which make up the core of the system. Two distributed information management applications have been written, a general query architecture and an open hypermedia application, 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 [10]; it is also essential that information suppliers are provided with tools that help them in authoring and maintaining it [4, 10].

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 [8]. 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 [31] or the numerous agents applied to the Internet or the WWW [6, 30].

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 [20], 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 [22], and the open hypermedia philosophy was brought to the WWW by the Distributed Link Service [4]. The same principles were also applied to other types of media, in particular to images [29] and sound [2]. In a project called Memoir [11], the notion of navigation trails was used to recommend documents that have been examined by user's sharing similar interests. These ideas were also applied to bookmarks, annotations and document ratings shared by users [14]. 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 [12] has been used to optimise queries of distributed information systems. Other DIM tasks have been investigated, such as link integrity maintenance [36] and authoring [4]. The benefit of mobility to solve distributed information management tasks was also studied [7].

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 [50], 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 *emerge* from 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 [50] 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 [27].

All coordinated activity of a multi-agent system is composed of agents with simple (usually singular) abilities who possess the above three criteria. The notion of *oppor-*

tunism enables us to build systems where agents can potentially discover new functionality through cooperation – emergent behaviour, in short. 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 [28]. By making use of other agents whenever possible, the whole is greater that 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 two 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 paper is to describe the SoFAR framework (SOuthampton Framework for Agent Research), its properties, its design and implementation, and to present our first agent applications built using the framework.

The key contribution of SoFAR is to apply some successful ideas of the distributed computing community to multi-agent systems. We adopt the same communication paradigm as Nexus [19], which is the communication layer that is used to build the Computational Grid [18]. 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.

This paper 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 (Section 4). In Section 5, we describe a general querying architecture and an open-hypermedia application that we have implemented with this framework. Finally, we discuss related (Section 6) and future work (Section 7).

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 coordinate their activity in order to provide an emergent behaviour. We believe that advanced behaviours can emerge from individual agents opportunistically exploiting services offered by other agents; it is therefore the goal of the framework to facilitate the reexploitation of these services. 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 is potentially a large number of agents that must be coordinated by the agent framework. The framework must be *light-weight* and *scalable*. By light-weight, 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. It is a requirement of the framework to be scalable: we want to avoid centralised components which would create bottlenecks during execution.

4. The communication primitives provided by the framework must be *independent* of the actual means of communication. 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.

In our framework, it is not a requirement to be directly compliant with standard agent communication languages such as KQML [15] or FIPA [16]. 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 or 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 [45, 46].

Nexus [19, 35] is a distributed programming paradigm, available as a library, that provides the essence of a distributed object system and has inspired the model of communication used in SoFAR. Nexus is the communication layer used in the Globus projects (www.globus.org), the basis of the Computational Grid [18]. Nexus has been shown 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[32]. In addition, each endpoint is associated with a table of handlers, from which one handler, i.e. a *method* in object-oriented jargon, 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 [28] 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 [43]. 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 [44], have less than half a dozen, while the more complex, such as KQML or FIPA have more than twenty. We believe that a frugal but careful choice of performatives would allow 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 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; 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 [28] also note that while objects control their state, they do not control their behaviour. A method invocation is an irresistible request which the object 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.

Object systems have evolved from explicit message-passing to method or function calls that activate computations, potentially remotely. Doing so, messages are no longer regarded as first-class entities. However, in agent systems, a message expresses the intention of an agent within a specified context; it is useful to consider messages as *first-class*, so that we can store them in knowledge bases or we can match them against other messages.

Therefore, our model of agent communication is defined in terms of startpoints and

endpoints, communication context, first-class messages, 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 proper multicast, 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 [24].

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. We take our usage of the term "ontology" from the work of Guarino and Giaretta [24, 25], as a vocabulary used for describing an intensional semantic structure. This structure, or *conceptualisation*, captures the semantic relations of all possible worlds of a problem domain, rather than of a particular state of affairs (as is the case with Gruber's extensional definition of a conceptualisation in [23]).

Traditional agent systems indicate the ontology in which the content of a message is expressed by a name, including it with the other contextual information for the commu-

nication. For a compound content term which contains other terms, this is usually taken to mean that all the terms in the expression are of that ontology.

We believe that this is overly restrictive, and prevents the easy reuse of objects and concepts between ontologies. The problem domains modelled by ontologies need not be disjoint, and common concepts, such as a person or a document, may appear in more than one domain. Disallowing mixed-ontology expressions means that each ontology must implement such common terms anew, or that there must be inheritance between ontologies, which introduces a fresh set of complexities.

The bibliographic metadata domain, one of the more common DIM domains, often uses mixed-ontology expressions. A bibliographic record representing a book may have fields which are written in different schemas (ontologies); the subject description for one book might use the Library of Congress subject classification, while that for another might use Dewey Decimal. These records do not have a single ontology, but have an ontology for each component part.

We want mixed ontologies for the reusability of term definitions, but this can be problematic, because in the absence of any other constraints, a term from one ontology may contain a term from any other ontology. We need to restrict the terms which can be contained in a given term in order to maintain consistency and promote structure in the domain model. Therefore, our ontologies, based on a predicate logic, are implemented as Java classes standing for predicates, terms or atoms. The typing of Java instance variables helps maintain the consistency of domain ontologies by constraining the relations in the conceptualisation of the ontology. In addition, we have implemented unification-based pattern matching and simple constraint satisfaction on the ontology terms to facilitate the processing of queries.

3.5 Architecture

SOFAR is written in Java, and all its agents are defined as a subclass of the abstract class "NullAgent". NullAgent provides communication initialisation, agent identification, and some utility functions for registering and finding agents. The agent framework contains a vital agent, namely the registry, used for brokering. DIM tasks may benefit from other predefined agents, such as the multimedia database, the WWW browser, the input and the output agents, which we present below.

Registry Every agent knows of one registry agent which it can use to advertise its services, or to find out about the services offered by other agents. These services are represented by a *capability* term which associates an agent with a (possibly non-ground) predicate on which that agent can answer queries. The registry acts as a *broker*, matching agents service needs to other agents which can provide those services.

Multimedia Database agent The multimedia database agent provides an interface to a database of metadata about a collection of media in an archive accessible via the WWW. The metadata stored is based on the Dublin Core element set [9], containing such fields as title, creator, subject, and format. The data is stored in a number of tables in an SQL database, which can be queried by agents.

WWW browser agent The WWW browser is an important tool in today's electronic desktop, and the browser activity reflects, in part, the user's activity. Therefore, the WWW browser agent exports browser activity information to the rest of the framework, by informing the agents who have subscribed to its service. Currently, the information is about visited URLs; tighter integration with the browser will allow us to provide more information, such as bookmarks or printed documents.

Input/Output Agents Some agents operating within the framework require inputs which need to be obtained from users, or produce outputs which are intended to be displayed as results. Rather than put the responsibility for interacting with the user on every agent in the framework that requires it, we have abstracted the interface away from the underlying processing.

Input Agent: This agent requests information from the user via dialogue boxes. A number of input agents can exist within the framework, taking responsibility for obtaining different types of information from the user. These might range from a simple text input dialogue, to a more complicated image browser that allows users to select images from a collection. Other acquisition mechanisms are also considered, such as speech or tactile interfaces.

Output Agent: This agent provides a facility for displaying objects, including URLs, hypertext, images, video or simply text. Such a component is being extended by a user's profile, maintaining how the user prefers data to be displayed (or printed, or said), what kind of format is required, etc.

In addition to the architectural components, several ontologies used in common DIM domains have also been defined in the framework with a view to providing a useful and flexible base set for developers. These include the following: *(i)* white pages directory information *(ii)* bibliographic metadata, supporting commonly used schemas such as the Dublin Core *(iii)* open hypermedia linking *(iv)* information on computer systems and their usage *(v)* representation of multimedia objects and processing *(vi)* WWW-related information, including user browsing records.

3.6 Contractual Registration and Subscription

Registration is the action by which an agent declares its ability to handle some messages (typically related to a specific topic) to the registry agent. 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 an act of subscription, it commits itself to honour such a subscription: whenever a fact changes it informs the interested subscribee. For each subscription act, a subscription manager issues a contract 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 order to facilitate information sharing, the registry was designed to help agents to advertise and find information. 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 of 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.

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. At the application level, agents such as the WWW browser agent, use the subscription mechanism to inform any agent that has declared its interest in a user's browsing activity.

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 opportunistically take 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 light-weight. 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 light-weight.

   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 [12] or hierarchical organisation [33] are being investigated.

4. Currently, our implementation relies on shared memory communications and Java RMI; we are planning to investigate the benefit of a lower-level communication layer such a Nexus, using the NexusRMI compiler [3]. Generic pretty-printers allow us to display terms of ontologies using KQML and FIPA syntax; we do not foresee any problem communicating these terms on the wire.

5 Applications to Distributed Information Management

Our ultimate goal is to build an advanced distributed information management system using the SOFAR framework we have presented. About thirty researchers of the Multimedia Research Group have taken part in two "three day sessions", which we call the "agentfest", with a goal to develop agents in SOFAR. As a result, two applications have been built to test the framework. Both applications help users to find information which satisfies their needs.

An advanced DIM system must be able to find an answer to any question, no matter how vague, by inferring its exact meaning from the user's activity, his/her profile, previously asked questions, in other words, from the user's *context*. Our long term goal is to get closer to such a system; in the meantime, we have taken two different approaches. The first application is a general query interface, which uses inputs from the user and some user model. The second application uses the open hypermedia philosophy to find relevant links to a multimedia document being viewed; these links can then be followed by the user.

In both cases, the applications demonstrate that our framework can coordinate the activities of many agents. Agents are able to opportunistically reuse information provided by other agents. The applications can be extended by adding new agents, whose services will be reused.

We are aware that a number of general query frameworks such as InfoSleuth [26] already exist. The purpose of the applications we describe here is to illustrate the suitability of SOFAR as a framework to coordinate DIM agents. In addition to the agents below, we have also written more than twenty agents for such tasks as audio contouring, content-based multimedia retrieval and network routing.

5.1 A Query Architecture

Figure 2 shows the agents that interoperate in the general query system we have built using SOFAR. The multimedia database agent, and the output and input agents were presented in Section 3.5; brief descriptions of the other components are given below.

Query Mediator Agent The query mediator agent coordinates the other agents, carrying out a single task, namely managing the information from the user's initial input through to displaying the final results.

Natural Language Query Resolver Agent The natural language (NL) Query Resolver Agent takes as input a NL query, and returns answers corresponding to that query.

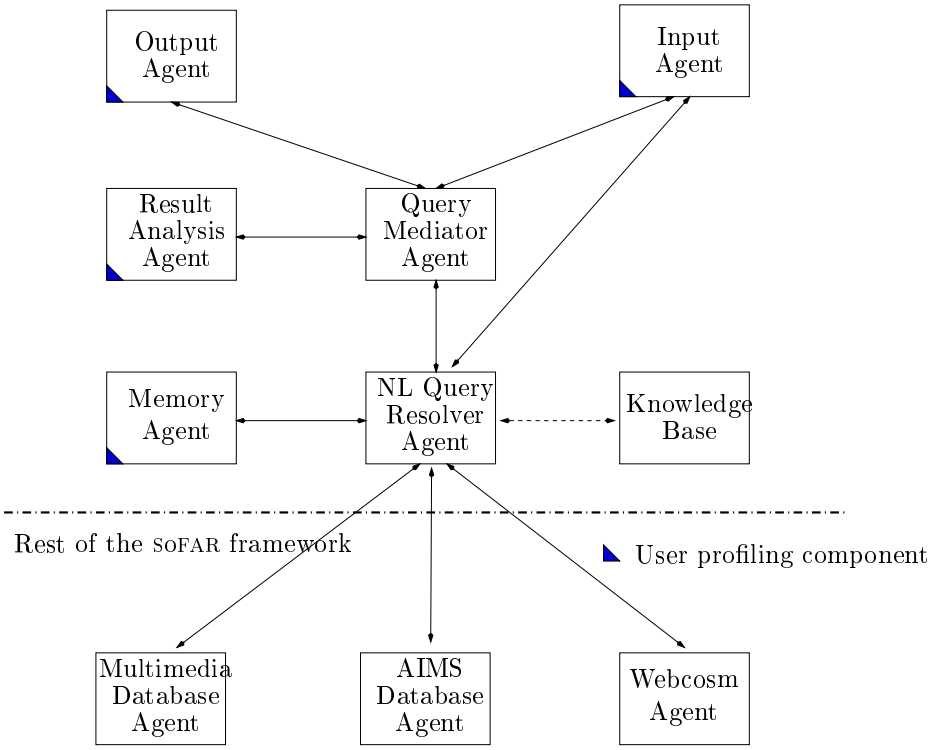

Figure 2: The interface agents within the SOFAR framework

The following two steps are carried out. First, a NL query is converted into a logical formula represented in our ontologies. Second, the Query Resolver finds agents able to answer queries based on that formula; these agents are queried, and results are returned to the mediator.

The NL Query Resolver agent uses a knowledge base that can be updated dynamically as new agents join the system. As agents register in the system, they can provide the Query Resolver with natural language queries or requests that they are capable of answering.

Results Analysis Agent Where a number of underlying queries are performed, there may be a need to amalgamate the results in some way before displaying them to the user. The Results Analysis agent receives a set of predicates and attempts to convert them to a more useful set of results for the user. The processing might include removing duplicates, ordering the results, or perhaps filtering based on criteria such as user preferences. The modular nature of the framework allows different Result Analysis agents to be plugged into the system. We are currently investigating how user feedback may be used to update user preferences, effectively building a user model.

An Example Query

To examine more closely how the agents interact, we trace a simple example query through the system. The mediator agent initiates the query by requesting the input of a text string query from the input agent. The text typed by the user is returned to the mediator, in

our example "Find information about SOFAR".

The mediator then asks the system for any agents that can take a text query and return answers to it. One of these agents is the NL Query Resolver agent, which is passed the query by the mediator. Using information in its knowledge base, it matches the user's query "Find information about" to the predicate SQLQuery. It thus instantiates this predicate with the text "SOFAR" which it extracted from the query string.

In this example, the predicate requires a minimum of one text string, which has been supplied as the parameter. If the user had only typed 'Find Information', the Query Resolver agent would call on the input agent to get the necessary parameters from the user to instantiate any missing variables in the query.

The NL Query Resolution agent now requests agents that can carry out the SQLQuery predicate. These agents appear in the bottom half of Figure 2 and are described below. The mediator calls these agents and receives a set of result predicates from both of them.

The mediator then finds an agent that performs result analysis, and sends it the set of results to combine and analyse; it receives back a processed set of results, which can finally be displayed by the output agent.

A number of stages in the process can benefit from user profiling information. To facilitate this, a *memory agent* keeps a record of queries and their corresponding results. The mediator calls the memory with a logical formula to see if it holds results for that formula in its memory. If it does and a similarity between the new query and the cached queries is satisfactory, then it can return the results; this similarity is computed by taking into account how important a term is in a query, and how well the term characterises the predicates. If it does not have any cached result, the memory agent responds negatively; then, the mediator calls an agent to process the query, also informing the memory agent of these results.

In fact, the memory agent acts as a model of the user, maintaining information resulting from previous queries which were returned to the user. As with all dynamic user models, issues concerning the consistency and the integrity of the stored information over time need to be addressed.

The input and output agents can also maintain a user's profile on how data should be acquired or presented to the user. In conjunction, with the result analysis, they help selecting and presenting relevant information to the user.

As shown in Figure 2, once the query has been composed, the NL Query Resolve agent sends queries to agents that support them. Several agents may be queried. First, the multimedia database agent (Section 3.5) provides access to a database of meta-data about multimedia documents.

Second is the AIMS (Academic Information Management System) agent, an interface to a WWW-based document management system providing access to a wide range of academic material. Documents are enhanced by a variety of open linking techniques based on the Distributed Link Service [4]. The AIMS agent provides full text search on the material held on the server and integrates legacy systems into the SOFAR framework.

Third, the WebCosm agent finds hypermedia links that are relevant to an input text. The agent makes use of WebCosm, a commercial implementation of the Distributed Link Service [4]. The agent can be thought of as a keyword finder and definition service for a particular subject area.

5.2 An Open Hypermedia Example

In open hypermedia architectures, information about the links between multimedia documents is stored and managed separately from the documents themselves, which remain in their native formats [20]. The links are objects in their own right; the link data consists of a set of associated *anchors* (e.g. locations in documents) and some related information. These links are kept in so-called *linkbases* and the corresponding servers are named *linkservers*. Adopting separate links has important advantages for authors and readers alike, especially when working with a complex, distributed information system [4, 20].

The Open Hypermedia Systems Working Group (OHSWG) [49] was set up to abstract the common features of various open hypermedia systems in order to achieve inter-operability. This effort resulted in *(i)* a common standardised data model for navigating multimedia documents, *(ii)* a reference architecture for Open Hypermedia Systems [41], and *(iii)* the Open Hypermedia Protocol (OHP) for communications in a component-based open hypermedia system.

The second application we describe in this paper demonstrates how the SOFAR framework can accommodate agents that exchange data based on the standardised linking model defined by the OHSWG. Basically two approaches can be distinguished: *(i)* building natively compliant agents from scratch, *(ii)* wrapping existing agents, in which we include the notion of *transducers* introduced in [21]. For reasons of proof of concept we have implemented both approaches. To this end, we define the OHP ontology, which comprehends the semantics of the OHP data model and operations, and which can be used by agents in the framework.

The design phase of the OHP within the OHSWG was dogged by issues that lay beyond the initial hypermedia concerns, such as protocol syntax, connection management and naming, pattern-based query handling and the management of a large set of processes. For this reason, we have found the use of the SOFAR framework for open hypermedia to be particularly attractive, as it has allowed us to concentrate on the modelling of hypermedia without being side-tracked by these other complex problems. Indeed, communications and naming are handled by SOFAR; message content must be part of some ontology and no syntax design is required, because data are serialised by the communication layer. Finally, the query language on ontology terms was more expressive than required by OHP.

Figure 3 presents the schematic organisation of the application, centered around the MediaApplet agent, which works as follows. On startup, it queries the system for agents supporting the OHP ontology (appearing in the bottom half of the figure). It then presents a popup menu with an entry for each of the agents it has found and lets the user select the agent to be queried by pressing a button. The results of the queries, presented as links that can be followed, are then shown by the MediaApplet agent.

When the MediaApplet agent wishes to know about "documents related to this one", it queries OHP-compliant agents for any anchors that may be found in the current document. When the user selects a link to follow, the MediaApplet agent sends a second query to retrieve the link structures which were associated with that anchor and subsequently retrieves the destination anchors and hence the documents that were related by the link.

While we envisaged that most OHP agents would be queried directly for this information it was also expected that some agents might wish to generate their links only once and then store them in a separate agent which would make that information available to the system. The *Link Server* agent component is just such an agent. It listens to

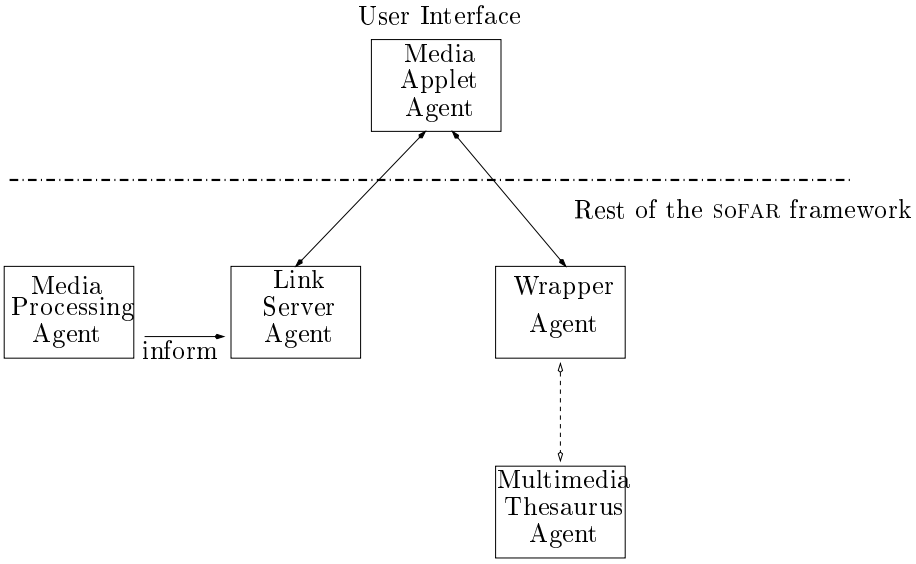

Figure 3: The MediaApplet

assertions made about link structures in the system and records them, retrieving them again when asked. For instance, in Figure 3 a media processing agent informs the link server about similarities between images.

Besides the use of native OHP agents, this application also shows how other existing agents can be integrated for purposes of information management. We have implemented a wrapper that maps some agents specific data structure onto a a dynamic link structure that can be interpreted by the MediaApplet agent just like standard link structures.

As illustrated in Figure 3, the wrapper is used to convert the semantic equivalence as defined by the *Multimedia Thesaurus* agent [13] into linking information. In essence, the multimedia thesaurus captures semantic knowledge about the relationships between media objects.

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 [28] for our view of agency, regarding it as a software engineering tool for managing the complexity of system development. Nwana and Ndumu [38] raise several points, namely that the standardised ACLscontain 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 [39], JAFMAS [5], JATlite [40], FIPA-OS [17], Ajanta [48] and JACK [1]. 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 [17] 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 to 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 [44] and the systems derived from it, such as AgentBuilder [42] or Becky Thomas's PLACA [47], 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* [44], but using KQML as its ACL rather than the much simpler Agent-0.

7 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 light-weight 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 thirty researchers in the Multimedia Research Group at Southampton (cf. Section 8). Training sessions were organised about agents, ontologies, and the actual framework implementation in Java. On two occasions, a group activity, called "agentfest", took place: during a three day session, those researchers developed agents, of which some were presented in this paper. As a result, SoFAR has now been adopted by several researchers for their everyday research, and eventually we envisage the framework being used for teaching. We welcome other researchers to try SoFAR, which is available from `www.sofar.ecs.soton.ac.uk`.

While numerous ongoing projects investigate the activity of distributed management using this framework, we concentrate here on future work related to the framework itself. First, mobility and ontologies for related concepts are to be made part of the framework. Our approach is based on an algorithm that transparently routes messages between mobile agents [34]. The communication model based on startpoints and endpoints will remain the same, but endpoints will be allowed to migrate with their associated agents. In addition, a model of distributed resources [37] will be integrated with SoFAR so that (mobile) agents are given the opportunity to reason about the resources they consume.

New modes of communication are 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, following the Nexus experience on this topic, security techniques such as encryption and authentication will be integrated in the communication model.

Once all the modes of communication we require become implemented, emphasis will be put on tools that facilitate the development of agent applications in SoFAR. In a first instance, an "ontology compiler" will convert an abstract specification of an ontology into concrete Java class definitions. Higher-level protocols that foster the cooperation, coordination and negotiation of agents are needed in SoFAR; tools have to be defined to facilitate the definition of such protocols in the framework. We will also investigate the possibility of reusing tools developed in other frameworks, such as Zeus [39], for the graphical development of agents.

8 Acknowledgements

Thanks to Colin Bird, Steven Blackburn, Les Carr, Steven Chan, Mark Dobie, David Dupplaw, Pierre Geurts, Jon Griffiths, Stephen Harris, Ian Heath, Jessie Hey, Simon Kampa, Shakeel Khoja, Nick Lamb, Paul Lewis, Rui Marinheiro, Kirk Martinez, Kevin Page, Stephen Perry, Guillermo Power, Jonathan Read, Alistair Riddoch, Neil Ridgeway, Mark Thompson, Mark Toller and Gary Wills, who took part in the first two "agentfests". Thanks to Nick Jennings for his comments on the paper. We acknowledge the EPSRC for supporting the postgraduate students and research assistants who have contributed to SoFAR.

References

[1] Agent Oriented Software Pty. Ltd. *JACK Intelligent Agents User Guide*, 1999.

[2] Steven G. Blackburn and David C. DeRoure. A tool for content based navigation of music. In *Proceedings of ACM Multimedia '98*, pages 361–368, September 1998. ISBN: 1-58113-036-8.

[3] Fabian Breg and Dennis Gannon. Compiler support for an RMI implementation using NexusJava. Technical report, Indiana University, 1997.

[4] Les A. Carr, David C. DeRoure, Wendy Hall, and Gary J. Hill. 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, December 1995. Appears in World Wide Web Journal issue 1, ISBN 1-56592-169-0, ISSN 1085-2301.

[5] Deepika Chauhan. *JAFMAS: A Java-based Agent Framework for Multiagent Systems Development and Implementation.* PhD thesis, ECECS Department, University of Cincinnati, 1997.

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

[7] Jonathan Dale. *A Mobile Agent Architecture for Distributed Information Management.* PhD thesis, University of Southampton, January 1998.

[8] Jonathan Dale and David DeRoure. Towards a Framework for Developing Mobile Agents for Managing Distributed Information Resources. Technical Report M97/1, University of Southampton, February 1997.

[9] DCMI. Dublin Core metadata element set reference description. Proposed recommendation, Dublin Core Metadata Initiative, July 1999.

[10] David DeRoure, Wendy Hall, Hugh Davis, and Jonathan Dale. Agents for distributed multimedia information management. In *Pratical Application of Intelligent Agents and Multi-Agent Systems*, pages 91–102, London, UK, April 1996.

[11] David DeRoure, Wendy Hall, Sigi Reich, A. Pikrakis, Gary Hill, and M. Stairmand. An open framework for collaborative distributed information management. In *Seventh International World Wide Web Conference (WWW7), Brisbane, Australia*, volume 30, pages 624–625. Elsevier, April 1998. Published in Computer Networks and ISDN Systems.

[12] David C. DeRoure, Samhaa El-Beltagy, Nicholas M. Gibbins, Les A. Carr, and Wendy Hall. Integrating link resolution services using query routing. In *5th Workshop on Open Hypermedia Systems (OHS5)*, Darmstadt, Germany, February 1999.

[13] M. Dobie, R. Tansley, D. Joyce, M. Weal, P. Lewis, and W. Hall. A flexible architecture for content and concept based multimedia information exploration. In *Proceedings of the Challenge of Image Retrieval (CIR'99)*, February 1999.

[14] Samhaa El-Beltagy, David C. DeRoure, and Wendy Hall. 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, April 1999.

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

[16] FIPA: Foundation for Intelligent Physical Agents. `http://drogo.cselt.stet.it/fipa/`.

[17] FIPA-OS. `http://www.nortelnetworks.com/products/announcements/fipa`, 1999.

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

[19] Ian Foster, Carl Kesselman, and Steven Tuecke. The Nexus Approach to Integrating Multithreading and Communication. *Journal of Parallel and Distributed Computing*, 37:70–82, 1996.

[20] Andrew M. Fountain, Wendy Hall, Ian Heath, and Hugh C. Davis. MICROCOSM: An Open Model for Hypermedia With Dynamic Linking. In A. Rizk, N. Streitz, and J. André, editors, *Hypertext: Concepts, Systems and Applications (Proceedings of ECHT'90)*, pages 298–311. Cambridge University Press, 1990.

[21] Michael R. Genesereth and Steven P. Ketchpel. Software agents. *Communications of the ACM*, 37:48–53, July 1994.

[22] Stuart Goose, Jonathan Dale, Gary J. Hill, David C. DeRoure, and Wendy Hall. An Open Framework for Integrating Widely Distributed Hypermedia Resources. In *Third IEEE Conference on Multimedia Computing and Systems (ICMCS'96)*, Hiroshima, Japan,, 1996.

[23] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. Technical Report KSL-93-04, Knowledge Systems Laboratory, Stanford University, August 1993.

[24] Nicola Guarino. Formal ontology and information systems. In *Formal Ontology in Information Systems: Proceedings of FOIS'98*. IOS Press, 6-8 June 1998.

[25] Nicola Guarino and Pierdaniele Giaretta. Ontologies and knowledge bases: Towards a terminological clarification. In N.J.I. Mars, editor, *Towards Very Large Knowledge Bases*. IOS Press, 1995.

[26] N. Jacobs and R. Shea. The role of java in infosleuth: Agent-based exploitation of heterogeneous information resources. Mcc-insl-018-96, Microelectronics and Computer Technology Corporation, March 1996. Presented at the IntraNet96 Java Developers Conference.

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

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

[29] Paul H. Lewis, Jo Kuan, Steve Perry, Mark R. Dobie, Hugh C. Davis, and Wendy Hall. Content based navigation from images. *Journal of Electronic Imaging*, 7(2):275–281, April 1998.

[30] Henry Lieberman. Letizia: An agent that assists web browsing. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Montreal, Canada, August 1995.

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

[32] Danius Michaelides, Luc Moreau, and David DeRoure. A uniform approach to programming the world wide web. *Computer Systems Science and Engineering*, 2:69–91, 1999.

[33] Luc Moreau. Hierarchical Distributed Reference Counting. In *Proceedings of the First ACM SIGPLAN International Symposium on Memory Management (ISMM'98)*, pages 57–67, Vancouver, BC, Canada, October 1998.

[34] Luc Moreau. Distributed directory service and message router for mobile agents. Technical Report ECSTR M99/3, University of Southampton, 1999.

[35] Luc Moreau, David DeRoure, and Ian Foster. 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, August 1997. Springer-Verlag.

[36] Luc Moreau and Nicholas Gray. 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, March 1998.

[37] Luc Moreau and Christian Queinnec. 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, October 1997.

[38] Hyacinth Nwana and Divine Ndumu. A perspective on software agents research. *The Knowledge Engineering Review*, January 1999.

[39] Hyacinth Nwana, Divine Ndumu, Lyndon Lee, and Jaron Collis. Zeus: A tool-kit for building distributed multi-agent systems. *Applied Artifical Intelligence Journal*, 13(1):129–186, 1999.

[40] C. Petrie. Agent-based engineering, the web, and intelligence. *IEEE Expert*, December 1996.

[41] Siegfried Reich, Uffe K. Wiil, Peter J. Nürnberg, Hugh C. Davis, Kaj Grønbæk, Kenneth M. Anderson, David E. Millard, and Jörg M. Haake. Addressing Interoperability in Open Hypermedia: The Design of the Open Hypermedia Protocol. *New Review of Hypermedia and Multimedia*, 1999. Accepted for publication.

[42] Reticular Systems, Inc. *AgentBuilder: an Integrated Tookit for Constructing Intelligent Software Agents*, February 1999. Available from `http://www.agentbuilder.com/`.

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

[44] Yoav Shoham. Agent-oriented Programming. *Artificial Intelligence*, 60:51–92, 1993.

[45] Jon Siegel. *CORBA fundamentals and programming*. Wiley, 1996.

[46] Sun MicroSystems. Java Remote Method Invocation Specification, November 1996.

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

[48] Anand Tripathi, Neeran Karnik, Manish Vora, Tanvir Ahmed, and Ram D. Singh. Ajanta – A Mobile Agent Programming System. Technical Report TR98-016, Department of Computer Science, University of Minnesota, April 1999.

[49] Uffe Kock Wiil and Serge Demeyer, editors. *Proceedings of the 2nd Workshop on Open Hypermedia Systems, ACM Hypertext '96, Washington, D.C., March 16-20. Available as Report No. ICS-TR-96-10 from the Dept. of Information and Computer Science, University of California, Irvine*, 1996.

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