Agents for the Grid: a Comparison with Web Services (Part I: Transport Layer)

Luc Moreau
Department of Electronics and Computer Science
University of Southampton
SO17 1BJ Southampton UK
L.Moreau@ecs.soton.ac.uk

Abstract

The notion of agent has of late become popular in the Grid community, as exemplified by several workshops on the use of agents in the Grid. What are agents for the Grid? What is the difference between agents and Web-services? *These are questions that we address by describing a port of the* SOFAR *agent framework to Web services in the context of a bioinformatics Grid. In this first paper, we focus our discussion solely on issues at the transport layer. Through an agent communication language (ACL) and an abstract communication model, we have been able to define a generic API to communications, and are able to support multiple protocols, including the* XML *protocol, the transport mechanism of Web services. This approach facilitates the development of applications, makes our environment future-proof, and promotes the open-ness of our Grid architecture to third-party developers.*

1. Introduction

MYGRID [35] is a Grid [18] pilot project funded by the UK e-Science initiative [43]. Its purpose is to provide a collaborative and supportive environment that enables geographically distributed biologists to achieve research goals more effectively, while enabling their results to be used in developments elsewhere in the community. The MYGRID project aims to build a personalised problem-solving environment for the e-Scientist.

A key requirement of MYGRID is the design of an environment in which collaborative distributed bioinformatics applications may be developed. Our methodology for future proofing the environment is to design a *generic architecture* able to support multiple existing protocols, languages and standards, and which hopefully will be able to accommodate future developments.

The biological community has been particularly active in defining customised APIs for commonly used programming languages such as BioJava [4] and Bioperl [5], and for distributed computing technologies such as BioCORBA [3]. XML protocol and Web Services [46] are also becoming widely adopted in this community. It is the purpose of MYGRID to support all these technologies. Furthermore, MYGRID services will support and will be integrated with Grid middleware, e.g. Globus [17], to maintain links with development in the Grid community.

Our goal is to design an *abstract communication architecture* that we can map onto concrete communication technologies. Open-ness and ease of development are two driving forces of our undertaking: *(i)* For open-ness, we will specify protocols on the wire, i.e. the message format for each protocol supported, so that third-party developers may elect to use MYGRID services or make theirs available to MYGRID. *(ii)* For ease of development, we will provide programming APIs, which the programmer can use to develop new tools and applications in MYGRID. A specific aim of the programming API is to provide an abstraction layer that hides the specific implementation details of each communication protocol.

We have adopted *agent-based computing* as the paradigm underpinning our architectural design. Three complementary reasons motivate our choice: *(i)* Agents are a *software engineering* unit [26], typically larger than a class or module, which can encapsulate roles and goals, and which can be composed in order to build applications. *(ii)* Agents are able of *complex interactions*, then forming a multi-agent system [27]; typical interactions include negotiation and collaboration, which allow agents to adapt their behaviour to the prevailing environments. *(iii)* From a *communicative viewpoint* [15, 16], agents are entities that communicate using an *agent communication language* structured around a set of message types (performatives), message content schemas (ontologies) and message meta-information. The third reason, i.e the agent communication language, is the key principle underlying our abstract communication architecture. The other two reasons are long term motivations for using the agent paradigm, but we do not discuss them here; we refer the reader to publications such as [26, 27].

We have prototyped these ideas into SoFAR, the Southampton Framework for Agent Research [33], an agent-based communication infrastructure, initially targeted for distributed information management applications; it now serves as the foundation of the MYGRID communication infrastructure. The SoFAR communication layer was already successfully mapped onto several implementations of RMI (including a UDP based one) and SSL; we foresee no problem in porting it to CORBA. We have recently completed the mapping of the SoFAR agent communication layer onto XML protocol [46]. This paper discusses the differences between agents and Web Services from an implementation perspective.

Web Services generally refer to a three-layer system composed of: *(i)* a transport layer based on XML messaging (XML protocol, formerly known as Soap [46]); *(ii)* a registry mechanism allowing the advertising and discovery of services based on UDDI [42] and described by languages such as WSDL [45] or DAML-S [13]; *(iii)* Some workflow system such as XLANG [41] or WSFL [30].

It is frequent to hear questions such as "*Why use agents and not Web Services?*", "*What is the difference between Agents and Web Services?*". The purpose of this paper is to answer such questions from our practical experience of mapping SoFAR to Web services. This paper is the first step of our investigation, and focuses solely on the transport layer; in a second paper, we will discuss registry mechanisms.

This paper is organised as follows. First, we describe our abstract communication architecture based on an agent communication language (ACL) (Section 2). Then, we explain our implementation mapping the ACL onto XML Protocol (Section 3). We then compare our system with related work (Section 4) and conclude the paper with a discussion on the differences between agents and Web services.

2. Abstract Communication Architecture

Our primary motivation for adopting the agent paradigm for the MYGRID environment is the notion of an "agent communication language" (ACL), an abstract communication language that can be mapped onto concrete communication protocols, hence supporting our desire of an open and future-proof environment.

In combination with an ACL, we also support an abstract communication model, inspired by Nexus "virtual communication links" composed of startpoint and endpoint pairs [19]. An ACL and virtual links have allowed us to design a generic API, which facilitates the development of complex distributed applications.

In this Section, we present our ACL and its integration in our communication model in order to provide a generic API.

2.1. Agent Communication Language (ACL)

The idea of an "agent communication language" dates back from the DARPA knowledge sharing effort, which led to the design of KQML (Knowledge Query and Manipulation Language) [15], and was followed later by FIPA (Foundation for Intelligent Physical Agents) ACL [16]. Three elements compose an ACL, which we describe below.

Performatives In agent systems, it is common practice to separate intention from content in communicative acts, abstracting and classifying the former according to Searle's speech act theory [39]. An agent's communications are thereby structured and classified according to a predefined set of "message categorisations", usually referred to as *performatives*.

The number of different performatives varies between different ACLs. The most simple, such as Shoham's Agent-0 [40], have less than half a dozen, while the more complex, such as KQML or FIPA have more than twenty. We carefully chose performatives that 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 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 support common primitive interactions such as querying, notifying, requesting, subscribing, and advertising. They are a compromise between extremes, being chosen in order to avoid the complexity and communication cost that composition would entail in the most common scenarios.

Adopting a rigid set of performatives also offers additional benefits. Implementations may be able to offer efficient dispatch on messages, avoiding a further interpretation layer that would be required by a non-existing or unlimited message categorisation. Additionally, they allow us to provide a generic programming API, as we discuss in Section 2.3.

Ontology 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. An ACL requires agents to adopt an agreed vocabulary with a *shared understanding* of some domain that they can use in their communication [24]. Such a vocabulary is usually referred to as an *ontology*, and in this context can be seen as a message content schema.

At the communication level, we are only concerned with defining the vocabulary used in messages. Terms of such a vocabulary would typically refer (through a URI) to their semantic definition, which in turn could be used internally by agents to reason about such terms. Practically, terms of a vocabulary can be defined by a CORBA IDL or an XML schema; in Section 2.4, we will present the definition language for specifying terms of our vocabulary. On the other hand, semantic definitions could be written in more expressive ontology languages such as frames or DAML+OIL [2].

The dynamic and evolving nature of a Grid environment requires the support of multiple ontologies, designed, revised and published by independent developers, at different times. It is a requirement of an ACL to be able to accommodate these.

Meta-Information Performatives and ontologies as described above are only a slight abstraction above a remote procedure call. However, as opposed to method calls in object-oriented programming, communication acts are *not* considered as irresistible requests in agent-based systems. Agents must be able to discriminate between messages according to their internal state 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 or to discriminate between senders. In complex interaction protocols, agents need to be able to determine what conversation a message belongs to; a conversation may be regarded as a generalisation of a peer-to-peer session to n agents. This information is usually not available in object systems, but should definitely be made available in an agent system.

Such a communication context is defined by a notion of `Envelope`, which is specified in Figure 1.

2.2. Abstract Communication Model

An agent communication language can be mapped onto several communication layers, and therefore satisfies our need for open-ness: for each protocol, we may easily derive a specification of messages, which may be used by third-party developers to program applications interacting with MYGRID.

```
<term name="Envelope" extends="Term">
 <field type="AgentTerm"    name="sender"/>
 <field type="AgentTerm"    name="receiver"/>
 <field type="Time"         name="timestamp"/>
 <field type="Integer"      name="identifier"/>
 <field type="Integer"      name="reference"/>
 <field type="Conversation" name="conversation"/>
</term>
```

Figure 1. Meta Information of a Message

However, an agent communication language alone does not provide a generic API to communications, because e.g. performing a CORBA method call differs substantially from invoking a service through the Apache Soap implementation.

Therefore, we have adapted a key concept of the Nexus communication layer [19] to the world of agents. 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 a form of method activation to start computations on remote objects.

Communications between agents take place over a *virtual communication link*, identified by a *startpoint* and an *endpoint*. An *endpoint* identifies an agent's ability to receive messages using a specific communication protocol. An endpoint extracts messages from the communication link and passes them onto the agent. A *startpoint* is the other end of the communication link, from which messages get sent to an endpoint. Communications are initiated by calling one of the performatives on a startpoint; this results in the transfer of the data to the remote endpoint, which passes the message to the agent.

There may be several startpoints for a given agent, each acting as a representative of the agent at remote locations. Each startpoint is associated with an endpoint by a virtual link, and encapsulates a specific communication protocol.

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. Such mode of communication is only permitted for performatives accepting this semantics (typically those that do not return results).

2.3. Generic API

We now have all the ingredients necessary to build a generic programming interface to agent communications. In order to communicate with an agent, one needs to acquire a startpoint that represents the agent and that supports the communication protocol suitable for the prevailing circumstances. (How such a startpoint can be acquired is beyond the scope of this paper, but may rely on the use of a lookup service.)

The details of the API are of course dependent on its embedding in a specific programming language. If we adopt an object-oriented methodology (specifically the language Java), we obtain the following.

A same interface defines the client and server side of the communication act. A startpoint supports a method for each performative defined in the ACL. A communication is initiated by performing a method call on a startpoint, passing as argument the message content and the envelope. Figure 2 contains the interface `Agent` both supported by a startpoint and an agent object.

```
interface Agent {
  boolean query_if(Predicate p, Envelope e);
  Predicate[] query_ref(Predicate p, Envelope e);
  void inform(Predicate p, Envelope e);
  void uninform(Predicate p, Envelope e);
  Contract register(Action a, Envelope e);
  void unregister(Contract c, Envelope e);
  Contract subscribe(Predicate p, Envelope e);
  void unsubscribe(Contract c, Envelope e);
  void request(Action a, Envelope e);
}
```

Figure 2. Agent Interface

The effect of invoking a performative method on a startpoint is to package the method call up as a message and to transmit it to the endpoint, which deserialises it and passes it onto the agent. Startpoints and endpoints have a crucial role: startpoints define the different components of the communication context, such as time or sender; endpoints reconstruct the communication context and make it available to the agent.

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; it is then returned as a result of the method invocation on the startpoint.

2.4. Ontology Support

SoFAR ontologies are defined in an XML syntax, as illustrated by Figure 1. Concepts are composed of a number of fields, and may extend other concepts based on a principle of single inheritance. The root of the hierarchy is referred to as a `Term`. SoFAR also provides primitive concepts corresponding to programming languages primitive types such as `Integer`, `Float`, ... A concept of "polymorphic vector" is also supported to describe collections of concepts. An "*ontology compiler*" takes XML defininitions of ontologies and generates their representation as Java classes. Additionally, it generates a "visitor pattern" [22] for these data structures, which we use for defining customisable and extensible pretty printers to text, HTML or XML.

We have also defined notions of equality, matching and subsumption (i.e. reasoning over hierarchies) on ontological terms. These are implemented as Java methods that are available for use on Java representations of ontology definitions. Matching and subsumption have turned out to be a powerful feature of the system, because they provide the foundation of an ontology-based query language.

We do not anticipate any problem compiling our ontological definitions to CORBA IDL, and we have experimented with their translation into Jini templates [36].

3. Mapping to the XML Protocol

In this Section, we describe how we used the Apache Soap [1] implementation as a transport layer for SoFAR. Such an experience enables us to compare the agent model of communication with the Web services transport layer.

3.1. Architectural Design

Figure 3 presents the architecture of our implementation of SoFAR over XML protocol. At the top center of the picture, we find an HTTP server, hosting a Soap servlet able to handle incoming XML messages as prescribed by XML protocol. An agent will be seen as a Web service supporting a method for each performative.

Figure 3 also shows the path taken by an `inform` message sent by an agent A to an agent B. As explained in Section 2.2, communications between two agents take place over a virtual communication link, identified by a startpoint and an endpoint. Different communication links may be available between a pair of agents; the figure only shows a Soap-based communication link.

In order to send an `inform` message, agent A calls method `inform` on a Soap startpoint representing agent B. The startpoint contains the URL denoting the HTTP server acting as a Web-service provider for agent B. The startpoint establishes a connection over HTTP, which is handled by the Soap servlet. As a same HTTP server may be used to serve several agents, a local naming scheme is used by the HTTP server to designate the agent B. In the picture, `id` denotes agent B; such an `id` is contained in the startpoint and passed to the HTTP server, where a "demultiplexer" dispatches the request to the appropriate receiving agent. To this end, the demultiplexer makes use of a "Soap-registry" mapping ids onto agent endpoints. The configuration of the Soap server is defined in a "deployment.xml" file, which specifies the Java class (here DeMultiplexer) handling incoming requests.

When an agent B is created and declares its support of the Soap protocol, it is assigned a new id, and its presence is

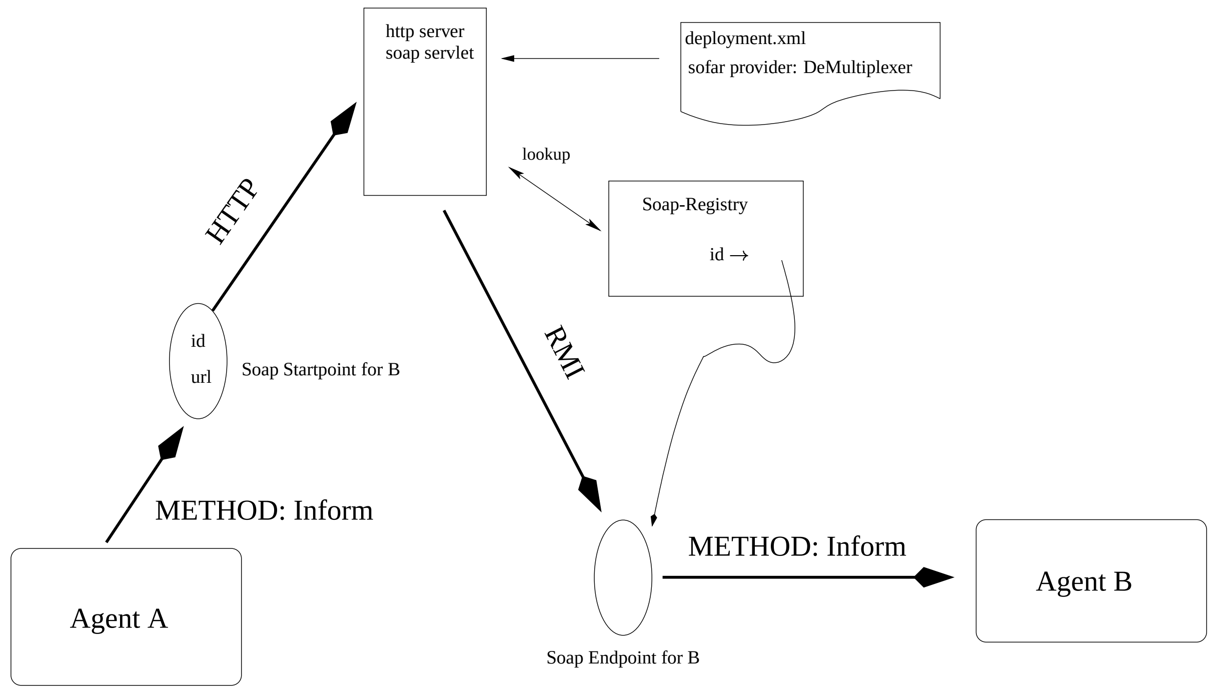

Figure 3. Communications of two Agents over SOAP

registered in a “Soap-registry”. The system is configurable in multiple ways. A same agent may be used as a back-end for several HTTP-servers, and vice-versa an HTTP-server may act as the Web-service front-end to several agents. An agent registered as a Web service can also support other forms of communication. For instance, outside an enterprise the agent may be seen as a Web service, whereas it is used with RMI inside the enterprise.

3.2. Technical Challenges

Various technical challenges had to be addressed while porting SoFAR to Apache Soap. Their understanding is useful for grasping the differences between the agent model of communication and the Soap approach.

Startpoints The envelope of a communication act (cf. Figure 1) contains a representation of the message sender, called an `AgentTerm`, which includes the startpoints the sender supports. In practice, we use RMI and Soap communications on a regular basis. An RMI startpoint encapsulates an RMI stub, which allows communication with a remote object using RMI. While the package Java RMI is able to serialise RMI stubs over RMI, it is currently unable to generate a serialisation of such a stub to XML. Furthermore, we cannot program their serialisation and deserialisation ourselves, because constructors for such objects are not public.

We therefore decided that only Soap startpoints would be serialised over the XML protocol transport layer. Their serialisation contains the URL representing the HTTP server and a unique identifier associated with an agent.

```
<SoapStartpoint url="http://..."
                id="..."/>
```

SoFAR is designed in such a way that constructors for startpoints and endpoints are not made public. The rationale is that a communication received from an agent running on a platform that is both authenticated and trusted to run the SoFAR system properly will correctly identify the sender. Such a rationale is no longer valid in the presence of Web services because the sender may simply open a socket and send XML data no longer guaranteeing any property. We are currently investigating the possibility of using digital signatures to authenticate message senders.

Besides the problem of security, transporting stubs over XML protocol breaks other properties such as liveness and distributed reference counting, which we discuss in the following paragraph.

Contracts RMI [25] and later Jini [36] have shown how the idea of a lease can be beneficial to clear registries (or lookup services). We have adapted this concept to the agent paradigm as follows.

Registration is the action by which an agent declares to an agent Registry 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 that ask matching queries. As a proof of its commitment, the registry issues a *contract* as a result of the registration act (as shown for performative `register` in Figure 2). 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.

Contracts are similar to leases as they need to be renewed on a regular basis by "keep-alive" messages to be sent to the registry. If the agent that registered or its host crash, "keep-alive" messages will no longer be sent, the lease will not be renewed, and the registry entry will be cleared. Furthermore, contracts may be shared and communicated by agents, which means that they need to be reference counted: a registry entry is cleared if no remote reference of the associated contract remains live in the system.

The SOFAR implementation of contracts relied on RMI stubs, for which the Java RMI implementation maintains reference counters and leases. Porting this aspect of SOFAR to Soap brings new challenging problems, because Soap (like Corba) opens the system to clients that may not be running Java. Hence, there is a burden on a client implementor to implement the lease and reference counting algorithms. Therefore, an application making use of these features, and involving a service not written in Java will have to trust the implementation of the algorithm supporting these features. This becomes particularly difficult when services are dynamically discovered and orchestrated. We may have to refer to some form of certification authority that certifies the validity of a given implementation.

In our implementation, a Soap Endpoint converts contracts into a representation that is suitable for XML serialisation. It is also its role to maintain the reference counter and lease associated with the contract; in addition, the HTTP server has to be ready to receive messages related to these. We are currently studying which distributed reference counting algorithm is the most suitable for a port over XML protocol. The issues to consider are the number of messages to be exchanged when contracts are communicated between agents, and the directions of these messages [32].

Class Loading Java RMI [25] extends Network Objects [6] by its ability to load code dynamically. Such facility is used by SOFAR to load the code of ontology terms dynamically, when received by agents. Let us remind the reader that this code is generated by the "ontology compiler" and provides important methods for matching terms and for visitor patterns used in a range of pretty printers.

The ability of loading this code dynamically is a fundamental property of a long-running multi-agent system. For instance, a registry agent may run permanently on a well-known host. As new agents and associated ontologies get developed, these agents will register their service in the registry. The registry will not be restarted with a new "class path", or recompiled with the new ontology; instead, it will have to load code dynamically.

On the other hand, XML protocol provides an RPC mechanism without a built-in dynamic code loading capability. It means that our port of SOFAR to XML protocol currently behaves properly for agents run with the knowledge of the ontologies supported by the agents they interact with. This is an unsatisfactory short term solution.

We are currently investigating a mechanism by which code could be loaded dynamically, based on the URI referring to the definition of an ontology. Since each ontology is required to be identified by a URI, the idea is to delegate to a third-party service the task of accessing ontologies definition from their URI, recompiling them on-the-fly and caching them. This third-party service could actually be the one hosting the definition of the ontology as specified by its URI. This scheme is however centralised and may not scale well in a Grid environment. Instead, we foresee several of these services replicated in the Grid, which would ensure that agents have the ability to load classes dynamically from a suitably-local and trusted service.

We have developed a very simple benchmark to evaluate the performance of SOFAR over Web services. An agent A discovers an agent B and sends 100 notifications using the `inform` performative. The SOFAR framework uses Java 1.3.1; we ran the benchmark with the Java native RMI implementation and with the Apache Soap 1.1 implementation (using Tomcat 3.2 servlets). We observed that the Soap communication layer was only 1.7 slower.

This result is positively surprising. Our implementation over Soap is still under development, and performs a Soap-registry lookup to find out the Soap endpoint and uses an RMI communication from the HTTP server to the Soap endpoint (cf. Figure 3). Our implementation could easily be optimised, for instance, if the Soap-registry was run in the servlet directly.

4. Related Work

A number of Grid systems make use of Web services as a communication layer. For instance, in Geodise, a Grid for engineering optimisation [12], it is shown how Condor can be offered as a Web service. The notion of agent has of late become popular in the Grid community, as exemplified by several workshops and publications on the use of agents in the Grid. Rana and Walker [38] advocate the use of the agent paradigm to integrate multiple information sources in problem solving environments. Busetta *et al.* [8] describe a Belief-Desire-Intention BDI agent architecture to simulate query optimisations in the Data Grid; in the long term, their goal is to provide advanced and adaptable Grid services (of which query optimisation is one) based on agent technologies. Rana and Moreau [37] review how agents techniques may be used to implement services at the computational Grid layer.

The DARPA CoABS (Control of Agent Based Systems) Grid [11] integrates heterogeneous agent-based systems, mobile agent systems, object-based and legacy systems. The CoABS Grid is based on Jini [36] for its lookup service and Java RMI for inter-agent communications. As such, it is less open than SoFAR, because it supports only one mode of communication. The CoABS Grid like SoFAR benefits from Jini leases to clear obsolete entries in registries. Test of scalability of the registration mechanism have been undertaken in [28].

While not mentioning agents explicitly, Furmento, Newhouse and Darlington [21] discuss another Jini-based technique for federating resources. Their long-term goal is the building of a computational economy for the Grid. Several other projects investigate this idea of a computational economy, according to which an economics framework regulates the supply and demand of resources. In particular, Nimrod/G [9] is a resource broker able of “budget-based” scheduling, giving users incentive to trade off execution time for economic cost.

The agent community is very active in devising high-level interaction protocols able to coordinate the activities of suppliers and consumers. Agents may interact using *co-operation* (cooperative problem solving), which is the process by which a group of agents choose to work together to achieve a common goal [44]. Multi-agent cooperation techniques are capable of adaptive behaviour. An alternative approach to this cooperation paradigm is the *market-based* model, where agents act as self-interested entities competing on a market, where goods such as computational resources are traded. Systems built around this paradigm have been shown to reach an overall equilibrium, in which resources are fairly allocated [10, 29, 31]. In particular, the market-based approach gives particularly good results when resources are becoming scarce [23]. The market-based approach is a specific case of a more general type of interaction among self-interested agents: *negotiation* [27]. The key characteristics of negotiation are: the presence of some form of conflict that must be resolved in a decentralised manner, by self-interested rational agents with incomplete information. Negotiation is the paradigm case of persuasion: it is a process by which agents come to a mutually acceptable decision on some matter. To the best of our knowledge, none of these techniques have yet been applied to the context of the Grid, though experience in resource allocation in communication networks, for instance, may be a good starting point for studying the techniques in the context of the Grid.

Tomarchio and Vita present a mobile agent architecture for monitoring services in a Grid environment. Our abstract communication architecture supports transparent communication with mobile agents, through the migration of endpoints [34]. We have not tested our Web service communication layer in this context yet.

The paradigm of agents has been used in the context of bioinformatics, but without specific concern for Grid environments. For instance, both [14] and [7] use agent systems to federate data sources and tools in bioinformatics applications.

5. Discussion and Conclusion

Both Web Services and agent-based computing are high-level models of distributed computing. Our port of SoFAR to XML protocols shows that: *(i)* Agents may be seen as Web services, and *(ii)* XML protocol is expressive enough to support an Agent Communication Language. Our port has also shown that an ACL is more abstract than XML protocol and can be mapped onto several concrete transport protocols. We note that XML protocol also supports different delivery mechanism (for instance HTTP and mail delivery), but message contents remains in an XML syntax, which is not directly in the philosophy of object-oriented systems such as CORBA. Additionally, a fixed set of performatives combined with an abstract communication model based on startpoint/endpoint pairs offers us a uniform API to programming communications. Importantly, this allows several transport protocols to be used simultaneously according to the delivery requirements of the application; for instance, we could conceive our SoFAR framework to be ported to a reliable group multicast system.

The choice of an ACL is also a move away from standard RPC because of the existence of the Envelope. From a service viewpoint, a client is no longer seen as an anonymous entity (or at most identified by an IP address), but as an identifiable agent with which communications are possible. Identifying service clients is an essential requirement for applications that need to trace provenance of data, or for applications required to provide services to entities belonging to a specific (virtual) organisation. The idea of a “current conversation” generalises peer-to-peer sessions to multiple-agent sessions.

Our integration of the agent paradigm and Web services will be pursued in different ways. This paper is the first part of a survey of agents and Web services. In a second paper, we will analyse the differences between agent registration mechanisms and Web service advertising using UDDI

(Universal Description, Discovery and Integration of Business of the Web) [42] and WSDL (Web Service Description Language) [45].

In the introduction, we discussed several motivations for using agents. We see two applications of agents' complex interactions for the Grid. *(i)* A computational economy may be a mechanism able to allocate resources in a fair and efficient manner in a Grid environment. This requires building a marketplace where rules enforce such a fairness property. We are currently designing such a marketplace approach for the dynamic selection of information in a recommender system; this system is built on SoFAR and we hope that some of these results will be applicable to a Grid context. *(ii)* The dynamic discovery, creation, management and disbanding of virtual organisations [20] require complex agent-based interactions; new scalable algorithms need to be devised in order to be applied to the Grid.

6. Acknowledgements

This research is funded in part by EPSRC myGrid project (reference GR/R67743/01) and QinetiQ and EPSRC Magnitude project (reference GR/N35816). The author thanks Carole Goble and Sean Bechhofer for a useful discussion on this topic.

References

[1] Apache SOAP. `http://xml.appache.org/soap/`, 2001.

[2] S. Bechhofer, C. Goble, and I. Horrocks. DAML+OIL is not enough. In *Semantic Web Working Symposium (SWWS-1)*, Stanford (CA), July 2001.

[3] BioCORBA project. http://biocorba.org/, 2001.

[4] BioJava. http://biojava.org/, 2001.

[5] Bioperl. http://bioperl.org/, 2001.

[6] A. Birrell, G. Nelson, S. Owicki, and E. Wobber. Network Objects. Technical Report 115, Digital Systems Research Center, Feb. 1994.

[7] K. Bryson, M. Luck, M. Joy, and D. Jones. Agent Interaction for Bioinformatics Data Management. *Applied Artificial Intelligence*, 2002.

[8] P. Busetta, M. Carman, L. Serafini, K. Stockinger, and F. Zini. Grid Query Optimisation in the Data Grid. Technical Report IRST 0109-01, Istituto Trentino di Cultura, September 2001.

[9] R. Buyya, J. Giddy, and D. Abramson. An economy grid architecture for service-oriented grid computing. In *10th IEEE International Heterogeneous Computing Workshop (HCW 2001), In conjunction with IPDPS 2001*, San Francisco, USA, Apr. 2001.

[10] S. H. Clearwater, editor. *Market-Based Control. A Paradigm for Distributed Resource Allocation*. World Scientifid Publishing, 1996.

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

[12] S. J. Cox, M. J. Fairman, G. Xue, J. L. Wason, and A. J. Keane. The Grid: Computational and Data Resource Sharing in Engineering Optimisation and Design Search. In *IEEE Proceedings of the 2001 ICPP Workshops*, pages 207–212, Valencia, Spain, Sept. 2001.

[13] DAML-S. `http://www.daml.org/services/daml-s/2001/05`, 2001.

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

[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] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *Intl J. Supercomputer Applications*, 11(2):115–128, 1997.

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

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

[20] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid. Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 2001.

[21] N. Furmento, S. Newhouse, and J. Darlington. Building Computational Communities from Federated Resources. In *Proceedings of the 7th International Euro-Par Conference (Euro-Par 2001)*, pages 855–863, Manchester, UK, Aug. 2001.

[22] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[23] M. A. Gibney and N. R. Jennings. Dynamic resource allocation by market-based routing in telecommunications networks. In *2nd Int. Workshop on Multi-Agent Systems and Telecommunications*, pages 102–117, 1998.

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

[25] *Java Remote Method Invocation Specification*, Nov. 1996.

[26] N. R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 2000.

[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] M. L. Kahn and C. D. T. Cicalese. CoABS Grid Scalability Experiments. In *Second International Workshop on Infrastructure for Scalable Multi-Agent systems at Autonomous Agents*, Montreal, Canada, May 2001.

[29] K. Kuwabara, T. Ishida, Y. Nishibe, and T. Suda. An Equilibratory Market-Based Approach for Distributed Resource Allocation and Its Applications to Communication Network Control. In *Market-Based Control. A Paradigm for Distributed Resource Allocation*, pages 53–73. World Scientific, 1996.

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

[31] M. Miller, D. Krieger, N. Hardy, C. Hibbert, and E. Tribble. An Automated Auction in ATM Network Bandwidth. In *Market-Based Control. A Paradigm for Distributed Resource Allocation*, pages 96–125. World Scientific, 1996.

[32] L. Moreau. Tree Rerooting in Distributed Garbage Collection: Implementation and Performance Evaluation. *Higher-Order and Symbolic Computation*, 14(4), 2002. (Coloured figures can be found in http://www.ecs.soton.ac.uk/ lavm/papers/hosc01-colour.tar.gz).

[33] L. Moreau, N. Gibbins, D. DeRoure, S. El-Beltagy, W. Hall, G. Hughes, D. Joyce, S. Kim, D. Michaelides, D. Millard, S. Reich, R. Tansley, and M. Weal. SoFAR with DIM Agents: An Agent Framework for Distributed Information Management. In *The Fifth International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents*, pages 369–388, Manchester, UK, Apr. 2000.

[34] L. Moreau and D. Ribbens. Mobile Objects in Java. *Scientific Programming*, 2002. Special issue of the International Workshop on Performance-oriented Application Development for Distributed Architectures (PADDA'2001).

[35] mygrid. http://www.mygrid.org.uk/, 2001.

[36] S. Oaks and H. Wong. *Jini In a Nutshell*. O'Reilly, 2000.

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

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

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

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

[41] S. Thatte. XLANG: Web Services for Business Process Design Author. Technical report, Microsoft, 2001.

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

[43] The UK Research Councils e-Science Core Programme. http://www.research-councils.ac.uk/escience/, 2001.

[44] M. J. Wooldridge and N. R. Jennings. Cooperative problem solving. *Journal of Logic and Computation*, 9(4):563–592, 1999.

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

[46] XML Protocol Activity. `http://www.w3.org/2000/xp`, 2000.