A Web Infrastructure for People and Agent Interaction and Collaboration

Giacomo Cabri, Letizia Leonardi, Franco Zambonelli
Dipartimento di Scienze dell'Ingegneria – Università di Modena
Via Campi 213/b – 41100 Modena – ITALY
E-mail: {giacomo.cabri, letizia.leonardi, franco.zambonelli}@unimo.it

Abstract

Internet pervasive services call for flexible supports to enable a wide degree of collaboration. On the one hand, many people are connected to the Internet and surf the Web not only to retrieve information, but also to carry out several kinds of different tasks via the on-line services. On the other hand, the Internet is likely to be soon populated by software agents that will act in behalf of users, "intelligent" enough to achieve the better result without boring their users. In this paper we present a Web-based modular architecture that permits interaction and collaboration among people and agents, leading to a more fruitful exploitation of the capabilities offered by the Internet. A negotiation application based on auctions is used to show the advantages of the proposed architecture.

Keywords*: Agents, Collaborative Work, Web Applications, Coordination, Auction*

1 Introduction

A lot of applications is being developed to exploit the positive features that characterize the Internet [Alm95]. Due to the characteristics of openness, heterogeneity and unreliability of the Internet, these kinds of applications call not only for tools and methodologies, but also for appropriate models and supports to make the design of the applications easier and to permit a fast spread of these technologies.

In this context, developers must take into account that the Internet world will be populated by heterogeneous kinds of entities, in particular *people* and *agents*. Many people discover how the use of the Web permits to save time even in performing simple tasks. Moreover, the Web give a world-scale spectrum of possibilities among which to choose the preferred one, letting people compare and make the best decision. Agents are proposing as other entities that will populate the Internet. They are software entities characterized by the following main features: *autonomy*, *proactiveness*, *reactivity* and *sociality* [JenW98]. While the first three features are related to the development of each single agent application (or of classes of applications), the fourth one has a deep impact in the development of Internet services. Moreover, we think that a further important feature for the agents is *mobility*, i.e., the capability of actively changing the hosting execution environment in a network-aware fashion [KarT98].

Even if both people and agents exploit the same Internet services and resources, the ways they do it can be very different, because of the differences between the adopted interfaces, the speeds of elaboration, the intelligence degrees, and so on. This calls for an adequate support to permit heterogeneous interactions and collaborations on the Web [BenHT97, Cia98]. In a previous work, we identified the requirements of a flexible architecture that could support collaborative work on the Web, and we developed PROOF [CabLZ99], a modular framework that meets such requirements. We now point out that such architecture is adequate for a people-populated Web, but has several limitations when it has to be exploited also in an agent-populated environment. In this paper we propose a more general architecture, designed starting from the previous one, which has been extended to support the hosting of (possible mobile) agents and to be enriched by high-level coordination capabilities. The resulting architecture, called PROOF v2, is very flexible and can be exploited by several kinds of Internet applications. In particular, in this paper we show the concrete use of such architecture in an auction-based Internet application, considered as a case-study.

The paper is organized as follows. Section 2 introduces the PROOF architecture and sketches its implementation. Section 3 shows how the PROOF architecture can be extended by adding mobile-agent and coordination capabilities. Section 4 presents an example of application

This work was supported by the Italian National Research Council (CNR) in the framework of the project "Global Applications in the Internet Area: models and programming environments".

that exploits the collaboration capabilities of the extended PROOF architecture v2. Section 5 reports the conclusions and the open issues.

2 The basic PROOF Architecture

The main aim of *PROOF* [CabLZ99] is to provide a mean to enrich the Web with computational capabilities for CSCW (Computer Supported Cooperative Work) without requiring significant modifications to current servers and clients. In fact, PROOF relies on the concept of *proxy server*, which stands in the middle between servers and clients. While traditional proxy servers are mainly used to provide cache functions, PROOF is much more flexible and can embody several different functionalities. Thus, a such kind of proxy server intrinsically becomes a workplace open to the Internet, where the cooperation between clients can occur without forcing clients to exit the workplace when accessing generic Web servers. Within our PROOF proxy server, any kind of computation can be enclosed, such as caching and dynamic production of result HTML pages, as in the server-side approach. Synchronous interactions between the clients and the proxy server can be enabled by letting PROOF insert specific applets into the pages that it provides to clients (browsers), thus enabling communication among people in the workgroup via the proxy server.

PROOF is based on a modular architecture, composed by a *framework* and several application *modules* (see Figure 1). The framework provides the basic proxy functionalities, such as the connections with the client browsers and with the Web servers, the user identification and authentication; moreover it permits the embodying of different application modules. Each module implements the behavior of one specific application. The framework can load one module a time.

PROOF is written in Java, to achieve portability. It exploits the mechanisms of *interfaces* and *reflection* to achieve modularity and an easy and dynamic installation of application modules into the framework.

The PROOF architecture is very general and it is not tightly bound to any specific application because it is based on the implementation of a framework that offers general-purpose application-independent functionalities. Different application modules can be implemented with a limited coding effort and easily installed within the proxy-framework.

3 Extending the basic PROOF Architecture – PROOF v2

The previously described implementation of the PROOF architecture is satisfactory for the use in a Web inhabited by people. But it could be more useful if it were exploitable also by software agents. In this direction, we present the extensions made to the PROOF architecture in order to be more open and better integrated in the Internet environment. Such extensions have been implemented in the second version of PROOF, *PROOF v2*. In particular, the two most significant extensions are related to *(i)* the capability of hosting mobile agents and *(ii)* the integration of an advanced coordination model to enable high-level interactions and synchronization.

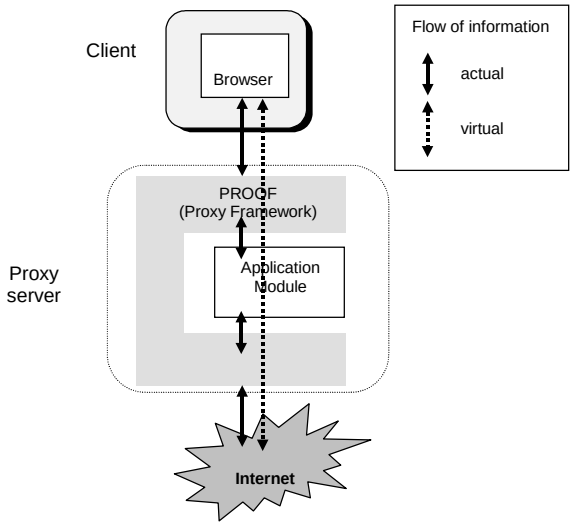

Figure 1. The application-dependent module defines the proxy server behavior

3.1 Adding Mobile-Agent Capabilities

The first important extension to the old PROOF architecture is the addition of the capability of hosting mobile agents. They can significantly improve the design and the development of Internet applications thanks to their characteristics. The *agency* feature [JenW98] permits them to exhibit a high degree of autonomy with regard to the users. The *mobility* feature [KarT98] takes several advantages in a wide and unreliable environment such as the Internet. First, mobile agents can significantly save bandwidth, by moving locally to the resources they need and by carrying the code to manage them. Moreover, mobile agents can deal with non-continuous network connection and, as a consequence, they intrinsically suit mobile computing systems.

Therefore, mobile agents, on behalf of users, can install specific modules to give the proxy server an application-dependent behavior. Since agents are autonomous, the user can give them a possibly high-level task to carry out, and

they can proactively search for the needed module(s), perhaps by mean of negotiation with module providers; once they have found the module(s), they can search for the most appropriate – also in terms of costs – proxy server where to install the module(s). The user is then notified of the proxy server (s)he have to use in order to exploit the needed functionalities. As a further advantage, this extension permits to give other entities – besides people – the access to collaborative Web applications. Users can think of not participating directly to an interactive application that requires repetitive actions; instead, they can rely on one or more software agents that act in behalf of them. For example, if a user is interested in buying a good, (s)he can delegate to a software agent the tasks of comparing the different offers, and of negotiating the final price. In this context, the exploitation of mobile agents can make the PROOF v2 architecture very open and flexible. The presence of different kinds of entities involved in the applications, introduces several interaction and synchronization issues, and calls for the presence of appropriate coordination models, detailed in the next Subsection.

3.2 Adding Advanced Coordination

Apart from hosting (possible mobile) agents, the PROOF v2 architecture also needs a sophisticated coordination model to permit interactions, collaborations, negotiations and also competitions at a high level. Since programmable tuple spaces are recognized to be a powerful coordination mean that well suits Internet applications, in particular those where mobile agents are involved [DenNO98], we have decided to extend PROOF with programmable tuple spaces.

The integration of coordination model based on programmable tuple spaces can be exploited also to store information about the proxy and the installed modules. This has a number of advantages. First, the tuple space is used as a shared repository where different entities can store, retrieve and exchange information in a simple and standard way. Second, the exploitation of programmable reactivity increases flexibility and permits to add whatever computation in the coordination media. For example, a reaction can be in charge of notifying a proxy module about a change in the tuple space; the proxy module, in its turn, can dynamically update the Web pages on the users' browsers accordingly to the new information in the tuple space (see Figure 2).

The concrete coordination system that we choose to be integrated in PROOF v2 is MARS (Mobile Agent Reactive Space) [CabLZ98], a coordination architecture based on an enhanced Linda-like model [AhuCG86]. MARS defines tuple spaces as shared repositories of information, used to store and associatively retrieve messages in the form of tuples (see Figure 2) [GelC92]. MARS was originally conceived for mobile agent applications, but it can be fruitfully exploited also by non mobile agents. The MARS architecture supposes that one tuple space is associated to each node.

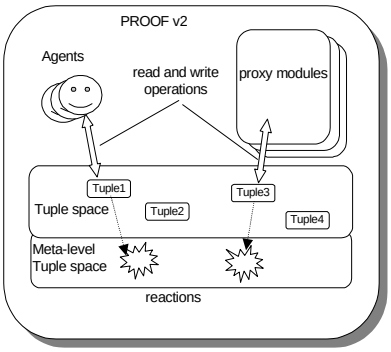

Figure 2. The MARS Architecture integrated in PROOF v2

MARS is JavaSpaces compliant and is written in pure Java. MARS tuples are Java objects that represent ordered sets of typed fields. Tuple classes must implement a specific interface required for tuple management and named Entry in JavaSpaces, or, in a simpler way, can sub-class from the so called AbstractEntry class, which provides for a basic implementation of the Entry interface. A specific tuple field is defined as an instance variable of a tuple class. Each field of the tuple refers to an object that can also represent primitive data. The MARS interface defines five Linda-like operations to access the tuple space:

- **write**, to put a tuple, supplied as parameter, in the space;
- **read**, to retrieve a tuple from the space, on the basis of a request tuple supplied as a parameter and to be used as a pattern for the matching mechanism;
- **take**, which works as the read operation but extracts the matching tuple from the space;
- **readAll**, to retrieve all matching tuples from the space;
- **takeAll**, to extract all matching tuples from the space.

MARS overcomes the limits of the static data-oriented Linda model, by adding programmable reactivity, which means that specific actions can be programmed in response to the accesses performed by agents on the tuple space [CabLZ00]. In fact, MARS implements a programmable reactive tuple space model in which the effects of the operations on the tuple spaces can be dynamically modified. Agents always access the tuple spaces with the same basic set of Linda-like operations. A meta-level tuple space is introduced in each node to associatively manage reactions. A meta-level tuple in the form (Reaction_object, T,

O, I) associates the reaction implemented by Reaction_object with the event of the operation O performed on the tuple T by the agent with identity I. Writing a meta-level tuple in the meta-level tuple space means installing the corresponding reaction, while taking a meta-level tuple means uninstalling the corresponding reaction. Readings on the meta-level tuple space are performed by the system to search for a matching reaction when an operation occurs on the base-level tuple space. Since T can be a template, a single meta-level tuple can be used to associate a reaction with the events related to all the tuples matching with T.

MARS can be exploited to implement agent-based applications in different areas. For instance, information retrieval, distributed management and mobile computing are the context where the MARS features can be exploited in a more effective way [CabLZ00c].

3.3 Implementation Issues

The two above-described extensions do not present hard implementation problems, since PROOF was originally designed taking into consideration modularity and flexibility. Moreover, the MARS coordination architecture has been designed to be very portable and uncoupled from Java-based hosting environments (usually, the agent systems): this makes easier to embody MARS in PROOF. However, some light modifications to the proxy framework are needed in order to achieve the highest usability and to exploit all features. In particular, the most significant one is that PROOF v2 must keep a reference to the local MARS tuple space, and must provide it to the installed modules; other more significant modifications to PROOF are needed in order to accept mobile agent modules; the main ones – detailed in the following of this section – are related to: multi-module capability, and security.

The first significant modification concerns the number of modules that can be loaded concurrently. While the first version of PROOF allowed only one module at a time, the new *multi-module capability* permits one single proxy to be exploited by several clients (browsers or agents) for different applications. Obviously, this capability introduces more complexity in the handling of the incoming requests and the related responses. To deal with more than one module in the framework, an additional software level is needed. Such level is in charge of *demultiplexing* the incoming request to the appropriate application module, so that each connected browser has the correct responses. Moreover, we must take into account the possible coordination among different modules installed in the same proxy framework. While the framework provides a simple mean of communication in the form of a global environment, the use of the MARS tuple space is encouraged to exploit all the advantages of its uncoupled and programmable coordination open also to external mobile agents.

The second aspect requiring modifications is related to *security*. In fact, if agents are allowed to install their own modules, security mechanisms and policies must be provided to control such installations, to avoid agents installing malicious modules, which can compromise the whole system by issuing, for example, a *denial of service* attack. The basic security mechanisms must be provided by the mobile agent system, in terms of identification and authentication of the incoming agents. The information about an agent permit to know whether it comes from a trusted user/site and to decide whether to allow the installation of an application module or not. Anyway, we choose to limit the visibility of the proxy internal state/functions from the agents. Moreover, the access to the internal environment by the application modules must be ruled with care, to avoid that a module has full power on the proxy state that relates also to other modules. So the access to the internal state of the proxy can occur only via a well-defined set of methods, which limits the possibilities of manipulate internal variables and information structures.

4 Application Example

An example of application that can fruitfully exploit the PROOF v2 architecture is an auction manager. *Auctions* [Ago96] are interesting negotiation means where there are entities that make resources available and entities that are interested in using such resources. The former ones are usually called *sellers*, while the latter ones are called *bidders*. Normally, there is an intermediate entity, called *auctioneer*, which actually performs the negotiation. The price of the resources sold by sellers via an auction is not fixed, but it is dynamically determined by the interest of the bidders. The seller can set a *reserve price*, i.e., a price under which it does not want to sell the resource. Intelligent agents can spend time to negotiate the desired resources by using the auction mechanisms, which seem to well fit dynamic and heterogeneous environments. In fact, an auction has a high degree of adaptability, has not any *a priori* fixed price and allows dynamic attending by participants. There are several different forms of auction, depending on the number of participants, on the criteria with which the resources are assigned, and so on. We focus on the auctions with one seller and multiple bidders at a time, ruled by several mechanisms: for example, English, Dutch, first-price and Vickery [Ago96].

Currently, there are several implementations of auctions described in the literature. With regard to human people, auction-based markets are rapidly spreading over the Internet, exploiting the Web infrastructure to permit interested people to interact [WWW98]. With regard to the agent-based applications, auctions can be used both to sell goods and to deal with Web resources [SanH00]. In the

following of this section we show how the PROOF v2 architecture can be exploited to build an auction-based application, which permits both people and agents to interact in an Internet environment.

4.1 Implementing Auctions with PROOF

First of all, an appropriate *auction proxy module* has to be implemented and installed in PROOF v2 to deal with the auctions. Appropriate *MARS reactions* are in charge of exchanging information between the auction proxy module and the tuple space, realizing the interactions between agents and people. On the one hand, people can participate by using their usual browsers, which are set to use PROOF v2 as the proxy server. On the other hand, agents can attend the auctions exploiting the tuple space integrated within PROOF v2, where they can find needed information about the goods/resources on sale, such as the price, the current highest bid, the timeout, and so on. Therefore, in this context, the features of the MARS model integrated within the proxy server can be useful exploited in the implementation of the auction mechanisms:

- the Linda-like data-oriented approach permits to access the selling/buying services in a simple and uniform way;
- the programmability property allows to uncouple auction mechanisms, implemented via reactions, from auction policies decided by people or embodied in the agents.

These aspects make possible to rule the agent's behavior depending on the specific auction laws. This achieves the same purposes of the socially adopted conventions of the traditional auctions [NorSR98].

The actual implementation of Internet auctions by using PROOF v2 works as follows. Let us suppose the following case for explanation purpose: the seller is represented by a mobile agent, while the bidders are represented by both human people and mobile agents. The bidder people participate by means of standard browsers configured to use PROOF v2 as proxy server. At connection time, they choose the auction module installed in PROOF.

At the negotiation site, the seller agent writes a tuple that contains information about the good/resource it is going to sell (step *a* in Figure 3). This writing triggers a specific pre-installed reaction that acts as the auctioneer, i.e. it is in charge of managing the auction (step *a'* in Figure 3). Such reaction is also in charge of notifying the auction proxy module that an auction has started (step *a''* in Figure 3).

Once the proxy module and the bidder agents have read the tuple representing the good/resource on sale, people and agents can bid a price to buy the good/resource on sale, respectively by using the applet displayed in the users' browsers or by inserting tuples (step *b* in Figure 3).

In the people case, the control applet communicates the new bid to the proxy module, which is in charge of updating the information in the proxy server environment by writing a tuple in the tuple space.

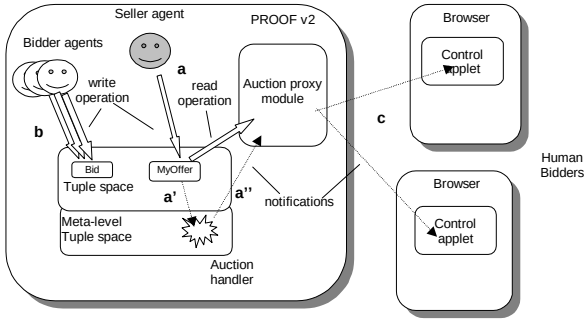

Figure 3. A seller agent puts a good on sale (a), the bidder agents put their bids (b), the auction proxy module updates the users' browsers via the control applets (c)

In the any case, the auctioneer reaction monitors the bids from both agents and people, and notifies the interested agents and the auction proxy module that a change has occurred in the auction. The module is then in charge of updating the information in the user browsers by reading the value of the new bid from the tuple space and communicating it to the control applets inserted in the auction Web page (step *c* in Figure 3).

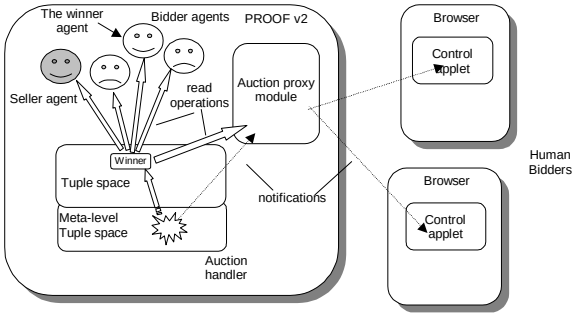

Figure 4. The Auctioneer reaction decides the winner and communicates it via a tuple

When the auction is over, the auctioneer reaction decides the winner and creates a tuple to inform all the participants about it: people are notified via the proxy module, while agents can directly read the tuple (see Figure 4). A detailed description of the implementation of auctions via reactions can be found in [CabLZ00b]. Obviously, the auction implementation must make use of security capabilities provided by the system to authorize only correct readings, takings and writings of tuples.

PROOF v2 permits to handle auctions in which both people and agents are involved, in a very flexible way. In fact, thanks to the modularity of the architecture and the

capability of programming the behavior of the tuple space, several kinds of different mechanisms can be implemented by changing (also at run time) the reactions installed in the system. For example, if a good is on sale by English auction rather than Dutch auction, the appropriate reaction can be installed to rule the interactions among participants.

5 Conclusions

In this paper we have presented the PROOF v2 architecture. Our aim is to realize a general-purpose architecture that supports the dynamicity, the heterogeneity and the openness of the applications in the Internet scenario. PROOF v2 is a modular architecture that well fits the Web environment, thanks to its "standing in the middle" between browsers and servers that permit to use existing Web components. The capability of dealing with mobile agents and the integration of an advanced coordination model permit to better exploit all the Internet features. On the one hand, the architecture becomes more open by letting also agents exploit its functionalities in terms of installing/uninstalling modules and interacting with the installed modules. On the other hand, the advanced coordination model based on programmable tuple spaces makes easier the design and the development of Internet applications, where collaboration, synchronization and competition have to be properly expressed and managed.

From the architecture point of view, PROOF v2 can be seen as a sophisticated interface to shared information. In fact, the information represented by tuples stored in the tuple space can be accessed by means of Internet browsers, which rely on an *ad hoc* proxy module. In this way, the module permits users to access and modify the content of the tuple space in an application-dependent way. Different modules can present a different vision of the information, depending on the specific application policies. As a final note, we point out that the presented new PROOF v2 architecture permit also to create a federation of proxy servers, which can be exploited in different ways. For example, the presence of several proxies with the same functionalities permits to perform load balancing by distributing the users towards the less loaded servers.

References

[Ago96] Agorics, Inc., "Going, going, gone! A survey of auction types", http://www.agorics.com, 1996

[AhuCG86] S. Ahuja, N. Carriero, D. Gelernter, "Linda and Friends", IEEE Computer, Vol. 19, No. 8, pp. 26-34, August 1986.

[Alm95] G. Almasi, A. Suvaiala, I. Muslea, C. Cascaval, T. Davis, V. Jagannathan, "Web*: A Technology to Make Information Available on the Web", Proceedings of the 4th IEEE Workshop on Enabling Technology: Infrastructure for Collaborative Enterprises, pp. 147-153, Berkley Springs (WV), IEEE Computer Society Press, 1995.

[BenHT97] R. Bentley, T. Horstmann, J. Trevor, "The World Wide Web as enabling technology for CSCW: The case of BSCW", in Computer-Supported Cooperative Work: Special issue on CSCW and the Web, Vol. 6, Kluwer Academic Press, 1997.

[CabLZ98] G. Cabri, L. Leonardi, F. Zambonelli, "Reactive Tuple Spaces for Mobile Agent Coordination", Proceedings of the 2nd International Workshop on Mobile Agents, Lecture Notes in Computer Science, No. 1477, Springer-Verlag (D), September 1998.

[CabLZ99] G. Cabri, L. Leonardi, F. Zambonelli, "A Proxy-based Framework to Support Synchronous Cooperation on the Web", Software, Practice and Experience, Vol. 29, No. 14, pp. 1241-1263, 1999.

[CabLZ00] G. Cabri, L. Leonardi, F. Zambonelli, "Mobile-Agent Coordination Models for Internet Applications", IEEE Computer Magazine, Vol. 33, No. 2, pp. 82-89, February 2000.

[CabLZ00b] G. Cabri, L. Leonardi, F. Zambonelli, "Auction-based Agent Negotiation via Programmable Tuple Spaces", 4th International Workshop on Cooperative Information Agents, LNCS, Boston (USA), July 2000, to appear.

[CabLZ00c] G. Cabri, L. Leonardi, F. Zambonelli, "Mobile Agent Coordination for Distributed Network Management", Journal of Network and Systems Management, to appear.

[Cia98] P. Ciancarini, R. Tolksdorf, F. Vitali, D. Rossi, A. Knoche, "Coordinating Multi-Agents Applications on the WWW: a Reference Architecture", IEEE Transactions on Software Engineering, Vol. 24, No. 8, May 1998.

[DenNO98] E. Denti, A. Natali, A. Omicini, "On the Expressive Power of a Language for Programmable Coordination Media", Proceedings of the ACM Symposium on Applied Computing, Atlanta (G), 1998.

[GelC92] D. Gelernter, N. Carriero, "Coordination Languages and Their Significance", Communications of the ACM, Vol. 35, No. 2, pp. 96-107, February 1992.

[JenW98] N. R. Jennings, M. Wooldridge, editors, "Agent Technology: Foundations, Applications, and Markets", Springer-Verlag, March 1998.

[KarT98] N. M. Karnik, A. R. Tripathi, "Design Issues in Mobile-Agent Programming Systems", IEEE Concurrency, Vol. 6, No. 3, pp. 52-61, July-September 1998.

[SanH00] T. Sandholm and Q. Huai, "*Nomad*: Mobile Agent System for an Internet-Based Auction House", IEEE Internet Computing, Special issue on Agent Technology and the Internet, 2000, to appear.

[WWW98] P. R. Wurman, M. P. Wellman, W. E. Walsh, "The Michigan Internet AuctionBot: A Configurable Auction Server for Human and Software Agents", 2nd International Conference on Autonomous Agents, May 1998.