Designing a document-centric coordination application over the Internet

Paolo Ciancarini and Davide Rossi and Fabio Vitali
Dipartimento di Scienze dell'Informazione
University of Bologna
Mura Anteo Zamboni, 7 - 40127 Bologna - Italy
E-mail: {ciancarini,rossi,vitali}@cs.unibo.it

Abstract

In this paper we describe an experience in designing a groupware application distributed over the WWW to solve a conference management problem. The system we design coordinates the activities of several people engaged in reviewing and selecting papers submitted for a scientific conference. We discuss why such an application is interesting and describe how we designed it. The architecture we suggest implements what we call an *active Web*, because it includes entities able to use and provide services offered through WWW infrastructures. Users, agents, and active documents can interoperate using a set of basic services for communication and synchronization. The active Web infrastructure we describe here is based on coordination technology integrated with Java.

Keywords:
Coordination, WWW architectures, Collaborative platforms, CSCW.

1 Introduction

In the current state, the World Wide Web does not provide enough support for document-centric applications based on agent-oriented ontologies, like group-

ware or workflow, which require sophisticated agent role-playing and coordination.

In fact, most WWW applications are either server-centric (the HTTP server connects to an application through the CGI protocol or any similar method), client-centric (applets providing application services to users without a real distribution of efforts), or not integrated at all with the Web (applications whose user interface is implemented by applets or plug-ins connecting with some proprietary protocol to a proprietary server, and using the browser as a passive and irrelevant host for the applet). All these approaches do not really satisfy the idea of an active Web based on some structured configuration of (autonomous) agents: they are either not really distributed, or not really integrated with the Web.

The World Wide Web is now the most popular platform to access Internet services, so it has the potential to become the standard infrastructure to build integrated applications. In fact, most application domains are turning to the World Wide Web as the environment of choice for building innovative applications leveraging on the open standards, their diffusion, and the programmable nature of the available services.

Among such domains, there is a growing interest in document management systems based on the WWW infrastructure. Some of these applications exploit *multiagent* technologies, meaning that they are highly concurrent, distributed, and often based on mobile code [MD97]. The PageSpace reference architecture [CTV+98] provides a reference framework for Web-based applications that are composed of autonomous agents performing their duties regardless of their physical positions. PageSpace provides clear-cut roles for agents, user interfaces and coordination paradigms in order to implement complex distributed applications on the World Wide Web.

In this paper we demonstrate the flexibility of PageSpace by describing an experience where this architecture has been tested. By relying on the coordination primitives of [CR97], a coordination language for Java, we have implemented MUDWeb, a cooperative interactive environment sharable by several users and autonomous agents remindful of the cooperative gaming environments called MUDs (Multi-User Dungeons). Within MUDWeb, a conference management

system has been implemented, allowing full support for all the activities connected to the organization of a scientific conference, such as the collection of papers, their distribution to reviewers, the collection of the reviews, and the selection of the accepted papers. ConfManager demonstrates the flexibility of the MUDWeb environment, which is a significant implementation of the PageSpace architecture.

The paper is structured as follows: in Sec. 2 we discuss the issues arising when creating distributed, interactive applications on the WWW. In Sec. 3 we discuss the case study of the design and implementation of a conference management system. We then introduce the PageSpace reference architecture in Sec. 4. Sec. 5 introduces MUDWeb: an instance of PageSpace based on Jada. After discussing the requirements in Sec. 6 we describe ConfManager, the actual implementation of the system based on MUDWeb. Section 7 concludes the paper and provides some hints on possible evolution of our research, in particular regarding the XML markup language.

2 The need for an active Web

The World Wide Web was originally born as a browsing system for hypertext documents working around a very simple set of common protocols and languages: the interactions among the WWW components are driven by the HTTP protocol, a very simple TCP/IP based client-server protocol used to retrieve documents stored under the control of an HTTP server. Often documents are in HTML format, a mark-up language based on SGML that allows the specification of hypertext links and simple formatting instructions for multimedia documents. An important characteristics of the WWW is that documents can be created on-the-fly by code activated by the HTTP server, as it happens with the CGI protocol, the servlet approach, or many server-side include languages such as PHP, Microsoft's ASP, etc. With these approaches a client requests a document as the output of a process run on the server. These simple mechanisms can be used in sophisticated ways in order to implement more complex forms of interaction among documents, clients, and servers.

According to its original design, the only activities that could be dynamically

triggered in the WWW were associated to one such server-side mechanism. Soon users demanded more interaction than just browsing documents, and this brought to the development of a family of languages that can be embedded into an HTML document and executed within the user's browser. These languages (or even architectures to transfer executable code, as in the case of Microsoft's ActiveX) are very different in capabilities and target; some in fact are scripting languages intended to interact heavily with the document itself (as in the case of JavaScript), while others are complex and full-fledged languages that have little interaction with the document (as in the case of Java).

These technologies give us the ability to "activate" two key components of the WWW architecture, servers and clients. However, we still lack standard and well-known techniques and protocols to allow these components to interoperate. Usually, in fact, projects aiming at the exploitation of the WWW as an active distributed platform locate computing components just on one side (either at the server or at the clients) or, if any more sophisticated need occurs, by inventing an *ad hoc* communication protocol between a specific client and a specific server-side application.

We propose a different approach to activate the Web, by redefining the coordination capabilities of the WWW middleware in which the activity takes place. An *active Web* is a system including some notion of agent performing some activity. These activities can take place at the client, at the server, at the middleware level, at the gateway with another active software system (e.g., an external database, a decision support system or an expert system) or even at the user level. An active Web includes several agents, all with well-defined behaviours. Each component of an active Web thus is an autonomous agent doing well-defined computations in a shared world providing coordination services: an agent is not just capable of computations but it also should be able to interact (in possibly complex ways) with other agents.

The interaction among agents is usually accomplished using client/server architectures (as in any RPC-based system, such as CORBA). However, the client-server framework misses its main goals (for instance, modular design and simple interaction behaviour) whenever the interactions among the components is unusually complex, for instance when the components change with time, when

the client/server relationship can be reversed, or when the designer needs a wider decoupling among components. These are issues we have to deal with in a world-wide distributed system including heterogeneous networks. A solution to these problems consists in designing the distributed application as a world of agents in which agents are spatially scattered and act autonomously: this schema fits quite well into the distributed objects model.

In a coordination-based approach [CarGel92], agents perform sequences of actions which are either method invocations or message deliveries. Synchronization actions (e.g., starting, blocking, unblocking, and terminating an activity) are the remaining mechanisms in object invocation. More precisely, we distinguish between *agent computation*, which is what concerns the internal behaviour of an agent, and *agent coordination*, which is what concerns the relationship between an agent and its environment, such as synchronization, communication, and service provision and usage.

Coordination models separate coordination from computation, not as independent or dual concepts, but as orthogonal ones: they are two dimensions both necessary to design agent worlds. A coordination language should thus combine two languages: one for coordination (the inter–agent actions) and one for computation (the intra–agent actions). The most famous example of coordination models is the one proposed in Linda [CarGel92], which has been implemented on several hardware architectures and combined with different programming languages. Linda can be seen as a sort of assembly coordination language in two ways. First and foremost, it offers very simple *coordinables* (i.e., active and passive tuples, which can be used to respectively represent agents and messages), a unique *coordination medium* (the Tuple Space, in which all tuples reside), and a small number of coordination *primitives*. Second, Linda is a sort of coordination assembly because it can be used to implement higher level coordination languages. For instance, we have used it to implement Jada, a coordination language for Java, briefly detailed in section 4.1.

Coordination models can be used to design an architecture that enables the active Web described in this section, a Web activities are not constrained to be either client-side or server-side. For instance, the PageSpace architecture [CTV+98] is a proposal for providing a general framework for active agents

within a Web environment that will be briefly detailed in section 4.

3 Designing a conference management system

As a case study for the usage of coordination technologies on the World Wide Web, we turned to a system to support the management of a scientific conference. The purpose of this section is to describe our experience: we intend to specify a system for supporting all the activities which typically have to be performed by a number of people widely distributed all over the world to submit, select, and prepare the set of papers which will be published in the proceedings of a scientific conference. The goal is to build the final version of conference proceedings, including a list of accepted papers.

This case study has been inspired by the personal experience of the authors of [CNT98] as PC members and especially by ideas described in [Sas96, MJ96, Nie97, Tol98]. In fact, there are already a number of Web-based conference management systems.

We have been inspired by some of these systems that provide support to electronic submissions, collection of referee reports, and PC meetings based on standard Internet technologies, such as e-mail and WWW.

In our experience the most robust and rich in features are the systems supporting the WWW and the AAAI conferences. Information on these systems is scarce; our description of informal requirements above is an attempt to summarize some existing systems. Interestingly all the systems we have examined support different conference organizational models. For instance, there are systems supporting conferences organized as a set of workshops; there are systems were papers are classified by authors according to some keyword systems, and then they are automatically assigned to PC members after they have selected a set of keywords representing their "reviewing ability". Most interestingly, only the simplest systems assume a centralized repository of papers and reviews. The system used for the WWW conference allows the authors to submit only the URL of their paper, which is then directly accessed by the reviewers only when necessary.

We are interested in using an agent-based software architecture. Further-

more, we will look for solutions supporting and coordinating both e-mail and WWW, in order to support both asynchronous and synchronous collaboration. Ideally, PC chairs, PC members and reviewers all have a reliable Internet connection, and work using a standard WWW browser. However, situations exist where this cannot be assumed: for instance, scientists usually travel a lot, and may have problems in having a persistent connection to the Internet. Moreover, most of the tasks to be performed are boring, long, and repetitive – e.g., fetching the papers, or filling referee report forms – and are better performed off-line.

3.1 Detailing the requirements

Following the idea that "coordination is the management of dependencies" [MC94], we can state that conference workflow management is concerned with managing the dependencies of activities (the workflow) necessary to produce the proceedings of a scientific conference (and of course the list of authors invited to present their papers).

Most scientific communities have established policies and mechanisms implementing some kind of conference management aiming at minimizing the organizational efforts but keeping high the quality of papers being accepted and the fairness of the selection process. It is a usual choice that a program committee is established to take decisions about which papers are accepted and which are rejected. Authors interested in presenting their work submit papers to such a committee for review. Within the committee, a group decision has to be taken according to some fair policy. The decision is usually reached by reaching a consensus or voting or ranking submissions with the help of several reviewers who help the PC members in evaluating each paper.

The system we intend to specify includes several agents. We take the term agent as primitive; intuitively, an agent is an entity which can act autonomously; an agent can send/receive messages according to some well-known protocol (non necessarily reliable): e.g., paper mail, e-mail, HTTP, or others. All agents have unique identities; for simplicity, we define these identities as unique URLs.

For instance, an agent is the conference site. Other agents can communicate with the conference site using HTTP and mail protocols. Some agents are

roles, namely they represent human users which can perform some operations: authors, reviewers, editors, PC chair, PC member.

The dynamics of the workflow is detailed in a number of tasks which the conference management system has to support. In some of these tasks there are some security and authentication issues at stake. The confidentiality of information needs to be ensured by using passwords and unique address for each instance of each role (PC Member, author, reviewer, etc.) to which any communication is directed. The tasks are listed below in some arbitrary order.

- Submission of papers: a submission form is available on request on the conference site. Authors submit papers to the conference site, attaching to the form all the required documents (including the paper, or the URL of the paper). The submission is immediately and automatically checked. If it is not conformant to the stated requirements, the sender is automatically invited to resubmit. If it is conformant it is stored in the conference site and the sender gets an acknowledgement.

- Bidding for papers. Each PC member examines the list of submissions and selects a subset of "interesting" papers to review. The PC chair can alter each subset to balance the review load among PC members.

- Distribution of the papers to the referees. After the deadline for submissions is expired, an agent sends by e-mail another agent to each PC member which, when run on the PC member's local system, fetches from a given URL all the papers assigned to that member for refereeing, and on request generates the forms for reviews to be sent by e-mail. Unauthorized access to all submitted papers has to be prevented, so that the list of the submissions itself – not to mention the papers – remains confidential.

- Collection of the reports. A WWW form is provided for online input of referee reports into the conference site. Also, forms can be obtained by sending an e-mail message to the conference site, and forms filled offline can be submitted directly by e-mail. This phase is managed by an agent that answers to requests of forms and collects the incoming reports. The

agent generates a document for each submitted paper in which it collects the relative reports.

- PC meeting. An agent scans all reports for quantitative values and fills up a table used to generate statistics and an initial ranking of all the papers. When the PC members meet, they need to be able to access the list, the abstracts, and the files of the submissions, the referee reports, the ranking table. Furthermore, they need to read or write comments about the papers, and reach a consensus on which papers to accept and reject. This is provided via (password protected) links to appropriate agents. In addition, all the features are available by e-mail, by sending messages with suitable subjects and content to the mail server agent. Appropriate steps need to be taken if PC members are allowed to submit papers, and thus need to the refrained from accessing the comments on their own submissions.

- Communication of results. Once the list of accepted papers is formed, an agent takes care of communicating the results and the referee reports to the authors. Other agents generate a list of the abstracts of the accepted papers, possibly extract a list of subreferees to evaluate the referee reports, and finally prepare a synopsis of the result for the PC Chair.

- Submission of camera-ready versions of the papers. Authors of accepted papers submit camera-ready versions of their papers to the site of the conference. An agent controls the compliance of the submissions with the standards for camera-ready submissions of the publisher. Furthermore, it warns late authors, if any exist. When all the accepted papers have arrived, an agent collects them all, plus a preface from the editor of proceedings, and prepares a draft of the proceedings. The editor and the PC chairs are informed by e-mail when the proceedings are done.

4 The PageSpace

PageSpace is a reference architecture for multiagent applications built on top of the World Wide Web [CTV+98]. Its purpose is to allow the deployment of

distributed applications using a coordination model the communication among the different modules, and the WWW as the underlying presentation platform. PageSpace applications consist of a number of distributed agents that have to be coordinated to accomplish some cooperative tasks.

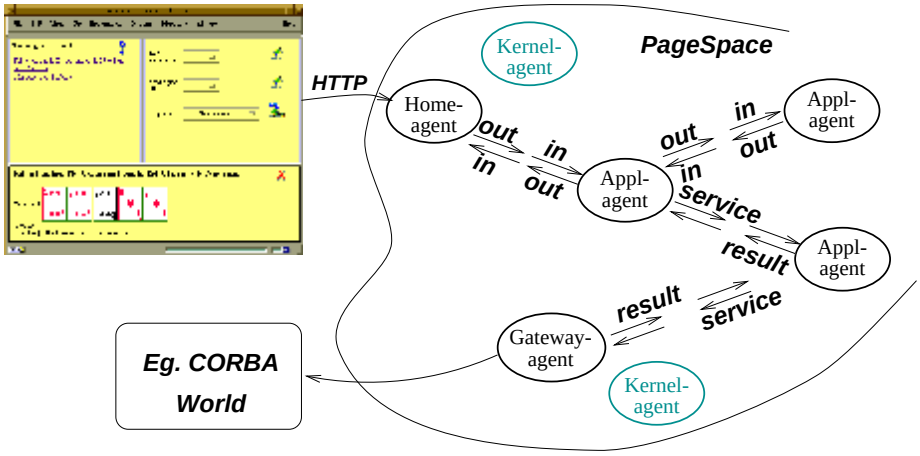

Figure 1: The PageSpace reference architecture

PageSpace includes a number of agent types, as shown in Fig. 1.

- *User-agents*, are user interface agents. User-agents consists of applets, scripts, and HTML pages.

- *Home agents* or *avatars*, are persistent representations of users on the PageSpace. They allow the users to access the application agents on the PageSpace by the mean of the appropriate user-agents, and collect the messages addressed to the user-agents in the ansence of the user. Home agents agents exploit the disconnected nature of the World Wide Web by allowing the graceful disappearance of remote entities of the cooperative tasks (such as human users and agents running on transient nodes of the network).

- *Application agents*, perform the computations of the applications. They offer services by interacting with the shared data. Some application agent have to interact with the user, and therefore must be able to produce an

interface, usually in the form of a user-agents that is then downloaded to the client. Others do not directly interact with the user, and just offer services to other application agents.

- *Coordination language* used for the specific implementation of a PageSpace. One such languages is Jada, described in section 4.1.
- *Gateway agents*, allow a PageSpace to interact with external environments such as other coordination environments or distributed middleware and applications residing outside the PageSpace.

Every browser includes at least one user-agent, which is connected and interacts directly with an avatar, running on a PageSpace server. A set of application agents implement the coordination mechanisms necessary to an active Web application. Gateway agents provide access to external services, like e-mail or a CORBA ORB.

The PageSpace reference architecture provides us with a generic framework to design and evaluate distributed applications on the Web.

Its main value, in our view, is that it provides an easily implementable solution for the main problems of the WWW in creating a distributed application, that is:

- the directionality of the HTTP protocol
- the possibility of all the parts acting as clients of the HTTP connection to retract or disappear for some time from the shared coordination environment

The concept of the avatar, a persistent representation of the disappearing agents that receives messages and in some ways act in place of the original agent, makes it possible to create a reliable distributed application even in the absence of some agents.

4.1 Jada

Jada [CR97] is a coordination language for Java. Using Jada we can coordinate parallel/distributed components using a set of operations introduced by the

coordination language Linda. The main idea is to have a basic set of primitives that allow the user to implement both data exchange and synchronization using shared, structured, data spaces.

Jada extends Linda's basic concepts replacing tuple spaces with object spaces (i.e. specialized object containers) and allowing the creation of multiple spaces.

The basic entity in Jada is a `Space`. Agents can access the space by using a small set of simple primitives:

- `out()`: put an object into the space;
- `read()`: associatively retrieve an object from the space; if more objects can be retrieved the one returned is chosen non deterministically; if no object is available at the time of the call the calling thread is blocked until the operation can be successfully performed;
- `in()`: works just like read but the returned object is removed from the space;

Non blocking and time-out aware versions of the `read()` and `in()` operations are also available.

The associative access to the object space performed by the `in()` and `read()` operations is explained in detail in Sec.5.1 where we will also see how Jada can provide a system based on PageSpace with a coordination kernel to deploy coordination-based multi-agent applications.

Note that after Java many other projects aiming at merging Java with various coordination technologies have been proposed. The most popular are probably JavaSpaces [JS99] and TSpaces [TS98] from SUN and IBM respectively, but, while the general concepts applied are the same, they present many differences in practical development and design.

5 MUDWeb

We decided to use a MUD environment for implementing the conference management system. We used MUDWeb, an actual software architecture designed

instantiating the PageSpace reference architecture on the MUD paradigm, and using Jada to provide the system with coordination facilities.

A MUD (Multi-User Dungeon) is a cooperative interactive environment shared by several people that use it to socialize and interact; MUDs have been proposed as enabling technologies for some kinds of groupware applications [ACMCSCW96, D$^+$97, DHW98]. A MUD usually represents the infrastructure of a role-playing game (hence the name) where human and robot players interact, visit dark and magical places, fight monsters or other players, and seek treasures. More generally, a MUD is an abstract platform that creates shared virtual realities. Thus, a MUD is a very powerful abstraction to describe a general platform for cooperative work (possibly on the WWW), that provides a general framework for users to interact with each other, and with resources such as documents.

MUDs are generally based on the concepts of rooms, items and players (or users). The whole virtual space inside a MUD is partitioned in rooms. Each room can contain several players and items. Each player can move from room to room, and can interact only with the items in the room he/she is in. Even interactions among users can take place only if the users are in the same room. In this context, the world room does not necessarily mean a closed place: a room in a MUD might virtually represent a cave in a dungeon or a garden around an enchanted castle. A room is simply a partition of the virtual space in which interactions take place. Players navigate through the rooms to interact, collect items and execute their goals. Players are not just humans: robot players behave just like other players but there is no human counterpart deciding their actual moves. The actions of the robot players are driven by (possibly intelligent) programs.

Mapping a MUD onto a coordination system based on multiple tuple spaces (like Jada) is quite straightforward: we can use a tuple space for each room; an item contained in a room is rendered as an object stored in an object space. Each player is an agent. Robot players are programs that access the MUD using the coordination primitives. We can use robot players to provide simple services to other users (in fact we will often refer to robot player as *server agents*). By interacting with a server agent, the users can activate a service and, eventually,

gather its output. Since the relationship among agents in Jada takes place by exchanging data, the same mechanism is used for all the exchanges within the MUD.

From a software architecture point of view, MUDWeb consists of a number of services which agents can use according to a number of protocols based on object exchanges. Services wait for command objects and perform services based on their content. Services are generally very simple and specialized agents that react to a limited list of commands. The functionalities of an application are thus implemented by a score of services cooperating among themselves.

Figure 2 shows the software architecture of MUDWeb.

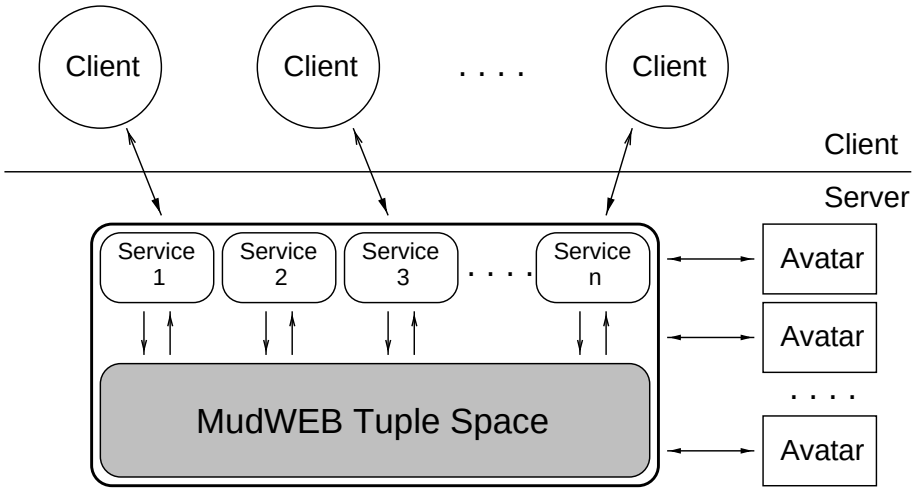

Figure 2: A MUD-like active Web

According to the PageSpace framework, we can identify three kinds of agents in MUDWeb: the avatars, the services, and the mudshell. The *mudshell* is the user agent, the interface framework where the interaction with the user takes place: it is a sophisticated HTML page with several widgets to provide commands the user to interact with the services by providing a MUD-like text box for direct commands, and displaying the most common commands on appropriate buttons. The *avatars* are the home agents, persistent representations of a human user. The avatars provide the user interface with commands that are displayed within a WWW browser, especially for moving from one shared space

to another, and can accept commands and return data in a variety of methods, including e-mail messages. The *services* are the application agents, the modules on the shared space that provide the actual computations of the distributed application.

5.1 Coordination in MUDWeb

The coordination facilities available for the agents running in MUDWeb are provided by MUDWeb itself. The implementation of the coordination facilities inside MUDWeb is based on Jada; we can then say that Jada act as a coordination kernel for MUDWeb. MUDWeb, like Linda, adopts tuple spaces for coordination purposes but, differently with respect to Linda, it supports the concept of *multiple nested tuple spaces*, which form a hierarchical coordination structure based on the `TupleSpace` object. Hence the `TupleSpace` object offers some methods to the agents to "navigate" through the coordination structure, and to express "itineraries" as sequences of names similar to UNIX paths.

After having created a `TupleSpace` object, an agent can connect to a tuple space with the method `join()`. For instance, after the following statement

```
TupleSpace ts = new TupleSpace();
ts.join("space1");
```

all operations on `ts` will be relative to the tuple space called "space1". Tuple space names can be specified with either a relative or an absolute name. It is also possible to move to the encompassing space by specifying the relative name ".." as the argument to the method `join()`. For additional flexibility, the method `leave()` is provided to move to the encompassing tuple space, and the method `leaveAll()` to move to the root tuple space.

A tuple is represented by the `Tuple` class and contains a set of Java objects. After having created a tuple, we can insert it in a tuple space with the method `out()`, as in the following:

```
Tuple alpha = new Tuple("Hello!", new Integer(1));
ts.out(alpha);
```

The methods `in()` and `read()` are used to retrieve tuples from tuple spaces.

```
Result read(Tuple formal);
Result in(Tuple formal) ;
```

Tuples are retrieved from a tuple space using an associative mechanism: when an agent calls the method `in()`, it has to provide as parameter a tuple to be used as a matching pattern. The method `in()` returns any one tuple (if any exists) that matches the given pattern. The same applies to the method `read()`, the only difference being that `in()` also removes the tuple from the tuple space.

Two tuples match if they include the same number of items and each item of the first tuple matches the corresponding item of the second tuple. In order to have a flexible matching operation we introduce the concepts of formal and actual tuple items: a formal item is an instance of the `Class` class (the meta-class used in Java). Any other object is an actual item.

Formal items are used to associatively access the contents of the tuple space by exposing a "template" of the items we are looking for. For instance, the following is a template tuple with an actual field accepting only the string "Hello", and a formal field accepting any integer:

```
Tuple template = new Tuple("Hello!", Integer.class);
```

The template tuple would then match both the tuples `alpha` and `beta`, but not `gamma`:

```
Tuple beta = new Tuple("Hello!", new Integer(3));
Tuple gamma = new Tuple("Hi!", new Integer(7));
```

Differently from Jada, disruptive MUDWeb operations do not directly return the matched item but a placeholder represented by an instance of the `Result` class. The placeholder can then be used to test the availability of a result, to fetch it or to kill the request. Trying to fetch a tuple that is not available will block the calling thread, thus giving us the same synchronization mechanism used in Linda.

Tuple spaces in MUDWeb can either be "local" (shared among concurrent threads running in the same Java Virtual Machine), or "remote" (contained on

a –possibly– remote host and accessed via a proxy class). The main feature of MUDWeb to support mobile agents coordination is the ability of transparently abort and resend a request for a pending `in()` or `read()` operation even from different network locations. Thus, if an agent performs an `in()` operation on a remote tuple space, and the requested tuple is not available at call time, the request can migrate to another place and the `Result` object will still refer to a valid `in()` operation performed on that remote tuple space. This feature provides the programmer with a tool to handle agent mobility and eases the implementation of distributed systems based on MUDWeb.

In addition to the previous basic tuple operations, MUDWeb introduces a new coordinative computing framework based on tuple collections. A tuple collection, represented by the `TupleCollection` class, defines a sequence of tuples having the same signature. In order to build a tuple collection we write

```
TupleSpace space = new TupleSpace();
Tuple pattern = new Tuple(String.class, Integer.class);
TupleCollection tc = new TupleCollection(space, pattern);
tc.add(new Tuple("Hello!", new Integer(1)));
```

where `space` is the tuple space where all the collected tuples reside and `pattern` is a formal tuple which defines the signature of the collected tuples. Actual tuples can then be inserted in a collection using the `add()` method. An exception is thrown when the tuple to be added has a different signature from the one of the collection.

The main feature of collections is that, through the use of iterator objects, the tuples of a collection can be read or withdrawn in the same order they were inserted. Two predefined iterators are provided, `ReadIterator` and `InIterator`, but more advanced iterators can be added to the framework, provided that they implement the method `nextTuple()`. For instance, the following code reads all tuples from the previous collection

```
TupleIterator iterator = tc.readIterator();
Tuple result;
```

```
while( ... ) {
        // get next tuple in this collection
        result = iterator.nextTuple();
        // use tuple items
         ...
}
```

The main feature of `ReadIterator` and `InIterator` is that the `nextTuple()` method is blocking. This way we can define an iterator that reads all tuples already inserted in the collection, but also all tuples that will be inserted in the future. The same result is obtained in other coordination languages such as Linda only by using an index as a tuple field; we believe iterators offer a more elegant solution.

Tuple collections and iterators capture a recurrent pattern of coordinative programming, the consummation of a sequence of tuples, and noticeably simplify the corresponding source code. Iterators are not built using a constructor, but with a factory method of the `TupleCollection` class. The main advantage is the possibility to develop extended collection classes which use the same factory method to create iterators. Iterators are also a well known design pattern [GHJV95] used to encapsulate access and traversal logic of an object container. Thus different iterators implements different access policies, leading to an improved design of the system.

To summarize: the coordination facilities available in MUDWeb are based on the Linda model (and are implemented using Jada as a coordination kernel) but extend it in various ways to better address the needs of open distributed collaborative applications like multiple distributed spaces, mobility support and bulk operations.

It should be noted that the differences between our system and other based on similar technologies but more focused on CSCW systems (such as the one presented in [MDP98]) have mostly to deal with the fact that we propose a generic infrastructure based on the MUD metaphore.

6 ConfManager

ConfManager is a conference management system based on MUDWeb that fulfills the requirements of Sec. 3.1.

To better understand the relationships between PageSpace, MUDWeb and ConfManager we recall that PageSpace is a reference architecture, MUDWeb is an actual software platform that instatiates PageSpace and ConfManager is an application built on top of MUDWeb.

In ConfManagers submitted papers are stored in rooms; authors, reviewers, and program committee members are all represented by avatars in order to support both synchronous (HTTP) and asynchronous (e-mail) interactions.

Other ad-hoc coordination-based conference management systems have been proposed (see, for example [Scu99]) but are not directly aimed for the WWW.

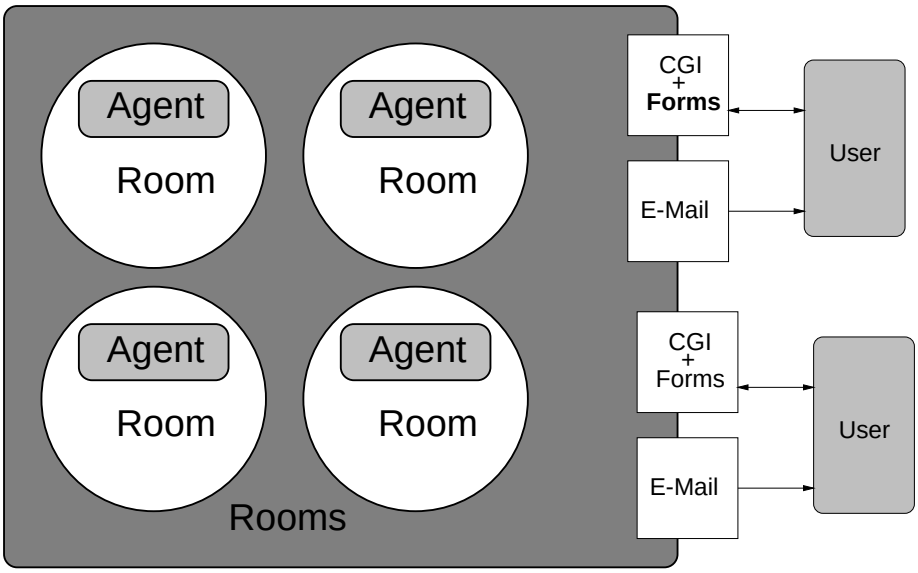

Figure 3: Conference management mapped onto MUDWeb

ConfManager includes the following rooms:

- SubmittedPaper Every paper is stored in one such room, that is dynamically created when the paper is submitted. The room will also store the reviews when they will be ready. The room also contains an avatar repre-

senting the corresponding author. The avatar can answer simple questions on the status of the submission.

- ReviewRoom is used by reviewers to store drafts of their reviews.
- SelectRoom is a room storing the scores assigned to papers. It is accessible to the program committee members only. The SubmittedPaper room temporarily contains the reviews for the paper it holds. After the reviews have been confirmed by the PC members, they are moved to this room.
- Papers is a room containing organizational data, such as the full list of submitted papers and the addresses of the authors. It is reserved to the conference organizers.

All services come in two flavors, synchronous and asynchronous. For instance, the synchronous service `Services.Submitter` accepts reviews coming from HTML forms displayed on the user's browser, while the asynchronous service `Services.Announcer` accepts reviews coming by e-mail. Asynchronous services rely upon avatars, which have to be programmed to perform the necessary tasks. For instance the PC Chair could program an avatar so that it forwards to his/her e-mail address each newly submitted paper.

Figure 4 shows the interface of a reviewer in the process of evaluating a paper. The interface is dynamically created by a MUDWeb agent that integrates with the HTTP server and act as a CGI entry-point.

While MUDWeb directly supports mobility we haven't addressed this feature in the actual architecture. Distributing services around a LAN is in fact possible but space and location issues must be dealt with (see, for example, [DRal00]).

7 Conclusions and Future Work

The integration of this kind of documents, that are active like agents, into the active items of a MUD environment based on PageSpace, is very easy and opens new and interesting opportunities to design a document management system based on the MUD metaphor, the PageSpace architecture, and a technology like Java. MUD's robot players can be seen either as synthetic users or as

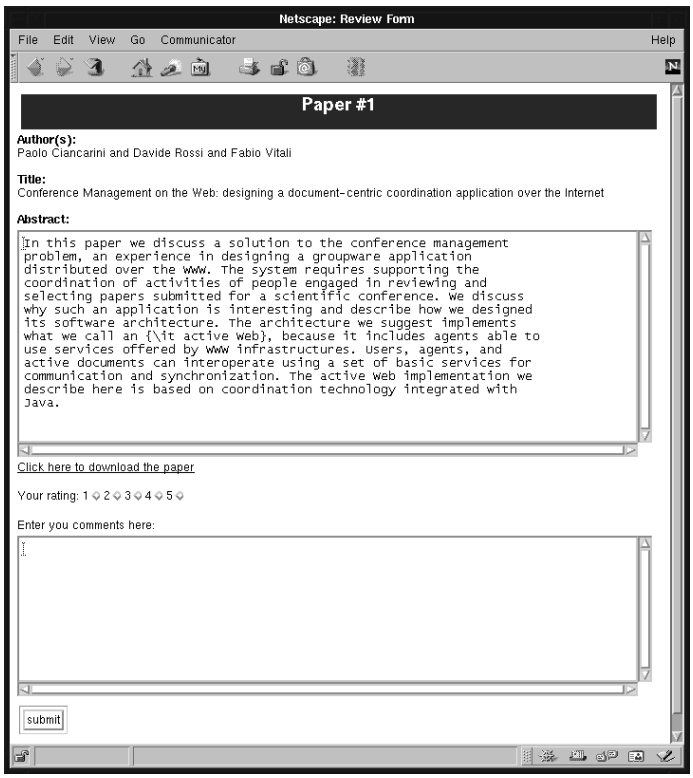

Figure 4: The reviewer interface

active items. We pursued only the first approach. It is however evident that there are classes of applications in which the documents we have to deal with can be really "active"; i.e. are subject to auto-modifications.

A document that represents a stock exchange's portfolio, for example, should periodically update its own value. A document that represents a contract should change its own state (and maybe also warn its owner) when its expiration date is reached. We refer to this class of documents as *active documents*. Inside a MUD the conceptual operation of mapping active documents into synthetic players is not natural and not correct: players and active documents are different concepts. It seems evident that active documents should be mapped into active items. The problem we face, using this approach, is that we need a standard framework that enable us to represent both the contents of the document and

its semantics.

XML [BPS97] is an extensible markup language that provides a unified framework for describing in an orthogonal way a document's content, structure, and rendering. Introducing a technique that we call "displets" [CVM99], our workgroup has integrated XML with a Turing-equivalent language, like Java or C-sharp, for manipulating the elements of an XML document. Displets allow us to produce active documents that can render themselves, or in general activate any arbitrary behaviour based on their content: we are planning to use this concept to implement some general-purpose active documents.

We have presented in this paper an experience in document-centric groupware. We know that several systems exist which support conference management; some of them work over the WWW or over proprietary platforms like Lotus Notes. We have sketched a solution using PageSpace, an agent-based reference architecture used to design an actual software architecture based on the MUD metaphor. The case study we have exposed is intended as a benchmark to compare modern object oriented middleware infrastructures: for instance, we are developing a similar conference management system based on Lotus Notes and in a future paper we intend to compare it with the present solution based on PageSpace.

We have already noted that an interesting feature of the case study is that there exist several "conference models", and that a truly flexible system should be able to support all of them. An issue that we have not discussed in this paper is what happens if documents to be managed are "active", i.e., when they include not only some contents but also some code. In fact, possibly the most interesting issue we are exploring is the integration in a coordination environment of active documents written in XML integrated with Java. We expect that coordination technology offers further degrees of integration and flexibility.

Acknowledgments. This work had partial support from a grant of Microsoft Research Europe and a grant MURST 40% project SALADIN.

References

[ACMCSCW96] Workshop Session of the ACM CSCW 1996. Design and use of MUDs for serious purposes. Proceedings of the ACM 1996 conference on on Computer Supported Cooperative Work

[BPS97] T. Bray and J. Paoli and C. Sperberg-McQueen. Extensible Markup Language (XML). The World Wide Web Journal, 2(4), 1997.

[CarGel92] N. Carriero and D. Gelernter. Coordination Languages and Their Significance. Communications of the ACM, 35(2), 1992.

[CNT98] P. Ciancarini and O. Niestrasz and R. Tolksdorf. A case study in coordination: Conference Management on the Internet. `ftp://cs.unibo.it/pub/cianca/coordina.ps.gz`

[CR97] P. Ciancarini and D. Rossi. Jada: Coordination and Communication for Java agents. In J. Vitek and C. Tschudin, editors, Mobile Object Systems: Towards the Programmable Internet, volume 1222 of Lecture Notes in Computer Science, pages 213–228. Springer-Verlag, Berlin, 1997.

[CTV+98] P. Ciancarini, R. Tolksdorf, F. Vitali, D. Rossi, and A. Knoche. Coordinating Multiagent Applications on the WWW: a Reference Architecture. IEEE Transactions on Software Engineering, 24(5):362–375, 1998.

[CVM99] P. Ciancarini and F. Vitali and C. Mascolo. Managing complex documents over the WWW: a case study for XML. IEEE Transactions on Knowledge and Data Engineering, 11(4):629-638, 1999.

[D+97] T. Das et al. Developing Social Virtual Worlds using NetEffect. In Proc. 6th IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), pages 148–154, Boston, June 1997. IEEE Computer Society Press.

[DS00] M. Divitini and C. Simone. Supporting different dimension of adaptability in workflow modeling. Computer Supported Cooperative Work, The Journal of Collaborative Computing, special issues on Adaptive Workflow, Vol. 9:3, 365-397, 2000.

[DRal00] Alan Dix, Tom Rodden, Nigel Davies, Jonathan Trevor, Adrian Friday and Kevin Palfreyman. Exploiting space and location as a design framework for interactive mobile systems. ACM Transactions on Computer-Human Interaction, Vol. 7:3, 285-321, 2000.

[DHW98] J. Doppke, D. Heimbigner, and A. Wolf. Software Process Modeling and Execution within Virtual Environments. ACM Transactions on Software Engineering and Methodology, 7(1):1–40, January 1998.

[GHJV95] E. Gamma and R. Helm and R. Johnson and J. Vlissides. Design Patterns Addison-Wesley, 1995

[MC94] T. Malone and K. Crowstone. The Interdisciplinary Study of Coordination. ACM Computing Surveys, 26(1):87–119, 1994.

[MDP98] M. Matskin, M. Divitini and S. Petersen. An Architecture for Multi-Agent Support in a Distributed Information Technology Application. Proceedings of the Workshop on Intelligent Agents in Information and Process Management, KI'98, TZI-Bericht no.9, pp. 47-58, 1998.

[MD97] J. Munson and P. Dewan. Sync: a Java Framework for Mobile Collaborative Applications. IEEE Computer, 30(6):59–66, 1997.

[MJ96] G. Mathews and B. Jacobs. Electronic Management of the Peer Review Process. Computer Networks and ISDN Systems, 28(7-11):1523, Nov. 1996.

[Nie97] O. Niestrasz. Identify the champion. `www.iam.unibe.ch/oscar/PDF/champion.fm.ps`, 1997.

[PDM] S. Petersen, M. Divitini, and M. Matskin. An agent-based approach to modelling Virtual Enterprises accepted for publication in International Journal of Production, Planning & Control, special issue on Enterprise Modeling, Taylor & Francis. (to appear).

[Sas96] V. Sassone. Management of electronic submission, refereeing, and PC meeting. (Manual of a WWW system), Nov. 1996.

[Scu99] A. Scutella. Simulation of Conference Management Using an Event-Driven Coordination Language. Proc. 3rd Int. Conf. on Coordination Models and Languages, Springer-Verlag LNCS Vol. 1594, 243–258, 1999.

[JS99] SUN Microsystems. JavaSpaces specifications. White paper, Jan. 1999.

[Tol98] R. Tolksdorf. Conference reviewing. `grunge.cs.tu-berlin.de/~tolk/reviewing.html`, 1998.

[TS98] P. Wyckoff and S. McLaughry and T. Lehman and D. Ford. T Spaces. IBM Systems Journal, 37:3, 454-474, 1998.