Internet-Based Coordination Environments and Document-Based Applications: a Case Study

Davide Rossi and Fabio Vitali

Dipartimento di Scienze dell'Informazione, University of Bologna - Italy
{Davide.Rossi, Fabio.Vitali}@CS.UniBO.IT

Abstract. In this paper we discuss a solution to the conference management problem, a case study in designing a groupware application distributed over the WWW. The case study requires supporting the coordination of activities of 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 its software architecture. The architecture we suggest implements what we call an *active Web*, because it includes agents able to use services offered by WWW infrastructures. A special kind of agents are *active documents*, which are documents that carry both some content and some code able to manipulate such a content. Users, agents, and active documents can interoperate using a set of basic services for communication and synchronization. The active Web implementation we describe here is based on coordination technology integrated with Java.

1 Introduction

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. For instance, there is a growing interest in document management systems based on the World Wide Web infrastructure. These applications typically exploit *multiagent* technologies, meaning that they are highly concurrent, distributed, and often based on mobile code [MD97]. We expect that document-centric applications will exploit the forthcoming XML technology, which allows to define *active documents*, that is messages which refer not only to content but also to code for manipulating such content [CV99].

However, the World Wide Web in its current state does not provide enough support for document-centric applications based on agent-oriented programming, like groupware or workflow, which require sophisticated agent coordination.

In fact, most WWW applications are either server-centric (interfacing applications via CGI to a mainframe-like central server machine), client-centric (applets providing application services to users without a real distribution concept), 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). 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 integrated with the Web.

The PageSpace architecture [CTV+98] provides a reference framework for overcoming these limitations, establishing an original design paradigm for Web-based applications that are composed of autonomous agents performing their duties regardless of their physical positions. In this paper we demonstrate the flexibility of PageSpace by describing a case study where this architecture has been fruitfully used.

The paper is structured as follows:

In Sect. 2 we describe a case study in document-centric groupware over the Internet: the management of a scientific conference. In Sect. 3 we discuss the issues arising when creating distributed, interactive applications on the WWW. In Sect. 4 we illustrate the PageSpace, a reference architecture for agent based interactive applications over the WWW. In Sect. 5 we discuss how we exploited PageSpace to design a conference management system. In Sect. 6 we conclude the paper and present some considerations on how to extend our approach to active documents.

2 The case study

The purpose of this section is to informally specify a case study that concerns the peer review and the evaluation of submissions to scientific conferences exploiting the Internet as the infrastructure for communication and cooperation. More precisely, 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 the list of accepted papers.

Following the idea that "*coordination is the management of dependencies*" [MC94], 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.

We believe that the proposed case study includes interesting coordination issues to study. It involves a set of coordination activities and requires coordination as a workflow itself. The case study can serve as an example of a real life application with prominent coordination aspects. The case study is sufficiently familiar to many scientists, who normally participate to conferences as either authors, reviewers, PC members, or PC chairs. The case study is also a simple instance of a general class of problems, in which a composite document has to be produced as the result of a workflow of activities by several people.

Interestingly, this case study is inspired by some already available systems for electronic submission, collection of referee reports, and PC meeting support based on Internet communication mechanisms, namely e-mail or WWW. In fact, there currently exist some conference management systems that use either the WWW or e-mail to perform most communication acts. Ideally, PC chairs, PC members and reviewers have a reliable Internet connection, and work in such an environment using a browser. However, scientists usually travel a lot. 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.

We are interested in solutions coordinating both communication media, that is, e-mail and WWW, in order to support both asynchronous and synchronous coordination.

2.1 Agents and Roles

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. snail 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 and is named `url.of.conference`. Another agent can communicate with the conference site using different protocols:

- `http://url.of.conference`
- `mailto://url.of.conference`

Some agents perform *roles*, since they represent human users which can perform some operations. The following roles are involved in the case study:

- author (submitter of a paper); if the paper is co-authored, the corresponding author is the author who actually submitted the paper;
- PC chair;
- PC member;

– reviewer;
– editor of proceedings.

Each role can be represented by several agents, which differ in their URLs. For instance, a typical conference could have a hundred or more authors, two PC chairs, twenty PC members, scores of reviewers, and one editor of proceedings.

The operations that each role can perform are those described in the next subsection.

2.2 Dynamics of the case study

We will specify the dynamics of the workflow to be enacted defining a number of tasks which the conference management system has to support. The tasks are listed below in some arbitrary order.

1. **SUBMISSION OF PAPERS**.
   A submission form, including basic information on the submission like authors, affiliations, paper title, corresponding author, format of the paper being submitted, is available on request from the conference site (authors can ask such a form to the conference site agent). Authors submit papers to the conference site.
   A submission may consist of the submission form appropriately filled, an ASCII file containing title and abstract and the paper (or its URL) in the format declared in the submission form (e.g. in PostScript).
   If a submission does not conform to the above requirements, it is not accepted and its sender is invited to resubmit. If a submission conforms it is stored in the conference site and its corresponding author gets an acknowledgement.
2. **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.
3. **DISTRIBUTION OF THE PAPERS TO THE REFEREES**.
   After the deadline for submissions is expired, an agent sends by email a message to each PC member to fetch the papers assigned to that member and start the refereeing. The message could even be "active", in that it would autonomously contact the site at the URL given, fetch the papers, and on request generate the forms for reviews to be submitted back to the conference site.
   Unauthorized access to all submitted papers has to be prevented, so that the list of the submissions – not to mention the papers – remains confidential.
4. **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 email message to the conference site, and forms filled offline can be submitted directly by email.
   In this and in the following phases there are some security and authentication issues at stake. The confidentiality of information has to be ensured by using

passwords to allow access only to PC members, and by using a unique address for each PC member to which all communication is directed.
This phase is managed by an agent that answers to requests of forms and collects the incoming reports. The agent generates a file for each submitted paper in which it collects the relative reports.
5. **PREPARATION OF STATISTICS** An agent scans all paper reports and fills up a table in a standard spreadsheet to generate statistics and an initial proposal for ranking all papers to be used during the PC meeting.
6. **PC MEETING.** During this phase all PC member needs to be able to:
   - access the list, the abstracts, and the files of the submissions;
   - access the referee reports collected so far for specific papers;
   - access the logs of comments on specific papers;
   - access the ranking table of all submissions;
   - read or write comments about a paper on a log file reserved for such task;
   - finally, decide on acceptance status and record the decision.

   All this is provided via (password protected) links from a WWW page to appropriate (active) elements. In addition, all the features are also available by email by sending messages with suitable subjects to the mail server agent.
   The situation gets more complex if PC members are allowed to submit papers. In this case one must deny them information about their own submissions, and, possibly, also the possibility of inferring such information.
7. **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 (for instance, in HTML), extract a list of sub-referees for the referee reports, and prepare a synopsis of the result for the PC Chair.
8. **SUBMITTING CAMERA READY**
   Authors of accepted papers submit camera ready versions. An agent controls compliance with the standards for camera ready submissions and issues reminders to authors that are late in re-submitting.
9. **PREPARING THE PROCEEDINGS**
   An agent collects all the camera ready papers, asks and waits for a preface from the editor of proceedings, and finally prepares a draft of the proceedings. The editor and the PC chairs are informed by e-mail when the proceedings are done.

2.3 "Solving" the case study

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 which are being used for supporting peer review. 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 instead allows the authors to submit *only the URL* of their paper, that is then directly accessed by reviewers only when necessary.

Here we are interested in "solving" this case study describing an agent-based software architecture of a conference management system.

3 Designing an active Web

The World Wide Web was born as a hypertext document browsing system. The components of this system are servers, clients, and documents. The interactions among these components are driven by the HTTP protocol and the CGI-BIN mechanism. HTTP is a simple interaction protocol used by clients in order to retrieve documents stored on a disk under the control of an HTTP server. Usually documents are stored as files on the server's disk. However, documents can also be created on-the-fly: with CGI-BIN a client requests a document as the output of a process run by the server at the client's request. Things are really as easy as that but this simple mechanism can be used in a "tricky" way in order to implement more complex forms of interaction among documents, clients, and servers.

According to its original design, the only activity that can be dynamically triggered in the World Wide Web is associated to the CGI-BIN mechanism. Soon users demanded more interaction than just browsing documents and this brought to the development of a family of languages to embed code within an HTML document and have it executed by 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), some are complex and full-fledged languages that have little interaction with the document (as in the case of Java).

Nevertheless, these technologies give us the ability to "activate" two key components of the WWW architecture, that is, servers and clients. However, we still lack techniques and protocols to allow these components to interoperate. Usually, in fact, current 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). Our approach is different because we redefine the coordination capabilities of WWW middleware in which the activity takes place.

4 The WWW as an agent world

An *active Web* is a system including some notion of agent performing some activity. The 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, decision support system or expert system) or even at the user level. An active Web includes several *agents*, with a well-defined behavior. Each component of an active Web is seen as an autonomous agent in a world offering coordination services to agents: thus, an agent is not just capable of computations but it should also 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, sometimes the client-server framework misses its main goals (that is, modular design and simple interaction behavior) when the interactions among the components change in time and the client/server relationship can be reversed, or when the designer needs decoupling among components (that is something we really need when we have to deal with a world-wide distributed system including heterogeneous networks).

A solution to this problem consists in designing the distributed application as an agent world in which agents are spatially scattered and act autonomously: this schema fits quite well into the distributed objects model. In our coordination-based approach 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*, that is what concerns the internal behavior of an agent, and *agent coordination*, that is what concerns the relationship between an agent and its environment: 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 programming models: one for coordination (the inter–agent actions) and one for computation (the intra–agent actions). The most known example of coordination models is the Tuple Space as available in Linda, 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 represent agents and messages, respectively; furthermore, it offers a unique coordination medium, the Tuple Space, in which all tuples reside, and a small number of coordination primitives. Secondly, 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 library which adds coordination primitives to Java [CR97].

4.1 The PageSpace

PageSpace is a reference architecture for multiagent applications built on top of the WWW [CTV+98]. PageSpace applications consist of a number of distributed agents to be coordinated to accomplish some cooperative task.

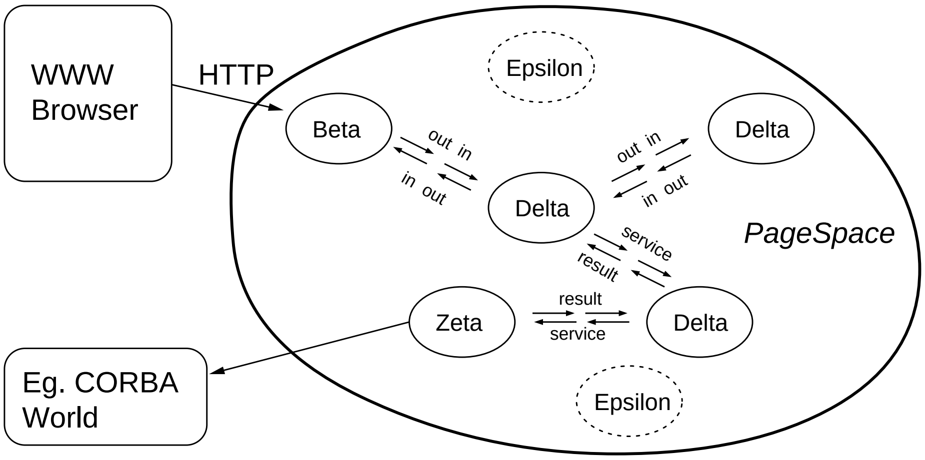

Fig. 1. The PageSpace reference architecture

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

- *Alpha* agents, or *user-agents*, are user interface agents. Alpha agents consists of applets, scripts, and HTML pages.
- *Beta* agents, or *avatars*, are a persistent representation of users on the PageSpace. They allow the users to access the application agents on the PageSpace, provide the user with the required Alpha, and collect the messages addressed to the Alpha agent or the user in their absence.
- *Delta* agents, or *application agents*, perform the computations of the application. They offer services by interacting with the shared data. Some delta agents have to interact with the user, and therefore must be able to produce an interface, usually in the form of an Alpha agent that is then downloaded to the client. Others do not directly interact with the user, and just offer services to other application agents.
- The *Gamma* space is a coordination environment used for the specific implementation of a PageSpace. In this paper we will use Jada [CR97].
- *Zeta* agents, or *gateway agents*, allow a PageSpace to interact with external environments such as other coordination environments or distributed 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 Delta 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.

4.2 Coordination mechanisms for agents

In [CR97] we introduced Jada, a Java-Linda combination. Jada was a first experiment in combining Java with a coordination model, and provides some basic coordination mechanisms. Here we describe an enhanced version of the original Jada: MJada.

MJada adds to Jada the support for coordination among (possibly mobile) agents. The MJada coordination medium is the tuple space, introduced in Linda [CarGel92]. However, Linda uses only one tuple space, although it can be physically distributed. On the contrary, MJada supports *multiple nested tuple spaces*, which form a hierarchical coordination structure based on the `TupleSpace` object. Hence a `TupleSpace` offers some "navigation" methods that allow a thread or an agent to navigate the coordination structure, expressing itineraries as sequences of names similar to `UNIX` file system names. Names are expressed by paths, so both `"/space"` and `"/chapter/section"` are valid names. A `TupleSpace` object can be created with

```
TupleSpace tuple_space = new TupleSpace();
```

An agent can connect a tuple space with the `join()` method. For instance, after the following statement

```
tuple_space.join("space1");
```

all operations on `tuple_space` will be relative to the tuple space called *space1*.

The `join()` method can be used to navigate through the tuple space as if it was a file system: absolute and relative paths are accepted, the name ".." refers to the encompassing tuple space, and name items can be combined (as in, for instance, `"space2/space3"`). For flexibility, additional methods are provided to move to the encompassing tuple space, `leave()`, or to the root tuple space, `leaveAll()`.

A tuple is represented by the `Tuple` class and contains a set of Java objects. We can create a tuple with standard Java constructors as in the following:

```
Tuple tuple = new Tuple("Hello!", new Integer(1));
```

Tuples can be inserted into a tuple space with the `out()` method.

```
tuple_space.out(tuple);
```

Tuples can be read or withdrawn from a space using the following methods

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

Tuples are retrieved from a tuple space using an associative mechanism: when an agent calls the `in` method it has to pass it a tuple that is used as a matching pattern. The `in` method returns a tuple (if any) that matches the given pattern. The same applies for the `read` method, with the exception that a tuple that is `read` is not removed 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 flexible matching operations we rely on the concepts of *formal* and *actual* tuple items: since a *formal item* is an instance of the `Class` class (the meta-class used in Java), all actual items of the specified class satisfy the matching.

For instance, the following is a template tuple with an actual field of class `String` and a formal field of class `Class` that matches all actual items of class `Integer`.

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

This tuple matches both:

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

Differently from Linda, disruptive MJada operations do not return a result tuple, but a placeholder for that tuple represented by the `Result` class. The placeholder can then be used to test result availability, to fetch a result or to kill the request. Trying to fetch a tuple that is not available will block the calling thread, resulting in the same coordination mechanism used in Linda.

Tuple spaces in MJada can either be "local", i.e., shared among concurrent threads running in the same Java Virtual Machine, or "remote", i.e., running on a (possibly) remote host and accessed via an ad-hoc proxy class in a way that is similar to the one used by RMI.

The main feature of MJada to support mobile agents coordination is the ability of transparently abort and re-send a request for a pending `in` or `read` operation among migrations. Thus, if an agent performs:

```
Result result=remote_tuple_space.In(my_tuple);
```

and the requested tuple is not available at call time, it can migrate to another place and the `result` object will still refer to a valid `in` operation performed on the remote tuple space.

MJada provides also multiple-result operations that allow one to read or withdrawn all tuples that match a given template.

- `Enumeration readAll(Tuple formal)`
- `Enumeration inAll(Tuple formal)`

Result tuples can then be fetched using Java enumerations. Linda does not have a similar operation.

In addition to the previous basic tuple operations Jada 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);
```

where `space` is the tuple space where collected tuples reside and `pattern` is a tuple which define the tuples' signature. Tuples can be inserted in a collection using the `add()` method

```
tc.add(new Tuple("Hello!", new Integer(1)));
```

If the specified tuple has a signature different from the collected one, an exception is thrown. The main feature of collections is that tuples can be read or withdrawn in the same order they were inserted.

Tuple collections capture a recurrent pattern of coordinative programming, the consuming of a sequence of tuples, and noticeably simplify the source code.

MJada has been used to implement an instance of PageSpace called MUD-Web.

5 Designing a conference management system

PageSpace is a reference architecture, independent from a specific coordination model or language. MUDWeb is an actual software architecture we designed instantiating PageSpace using MJada. MUDWeb takes its name from MUDs. A MUD (Multi User Dungeon) is a cooperative interactive environment shared by several people to socialize and interact; MUDs have been proposed as enabling technologies for some kinds of groupware applications [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.

The MUDs, generally, are 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 users and items. Each user can move from a room to another one can can interact only with the items in its same room: use can use them, get them and so on. ¿From this point of view objects are passive entities that are "activated" by the users. Also interactions among users can take place only if the users are in the same room (note that 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 a partition of the virtual space in which interactions take place).

As we pointed out, when wandering inside a MUD we might meet robot players. As the name suggests the robot player behave just like other users but there is no human counterpart behind it: the actions of the robot player are driven by (possibly A.I. based) programs. We can think of a robot player as a synthetic user or as an active item. We'll show later that these two concepts can lead to very different implementations.

Mapping a MUD in a system based on multiple tuple spaces like MJada is quite straightforward: we can use a tuple space for each room; an item contained in a room is a tuple stored in the tuple space representing the room; each user is an agent that can freely move around rooms.

Robot players are synthetic users, i.e., programs that access the MUD using the MJada primitives. We can use robot players to provide simple services to other users (in fact we will often refer to robot player as to 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 MJada takes place by exchanging tuples, the same protocol is used to activate the services of the server agents.

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 tuple exchanges. MUDWeb includes several rooms which correspond to tuple spaces including some basic server agent. The whole architecture is implemented by a server organized as nested tuple spaces. Fig.2 shows the software architecture of MUDWeb.

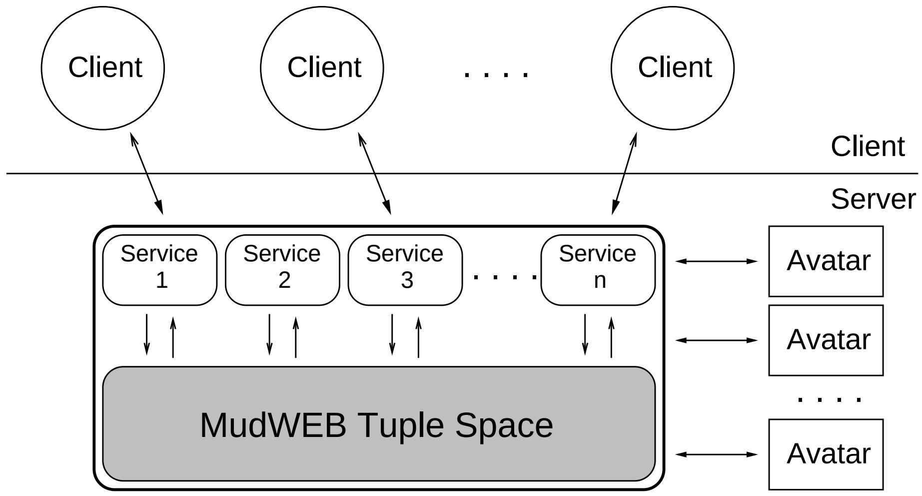

Fig. 2. A MUD-like active Web

Services wait for command tuples and perform services based on their content. Services are generally simple, specialized agents that react to a limited list of commands. The functionalities of an application are thus implemented by a number of services cooperating together. Clients are user interfaces using some role-specific HTML page or Java applet.

An agent server supports three kinds of agents: the *avatars*, the *services*, and the *mudshell*. The avatar is the persistent representation of a human user. The avatar interacts with the user interface displayed within a WWW browser, and it accepts commands and returns data in a variety of methods, including e-mail messages. Services are the PageSpace's Delta or application agents, i.e., they are the modules on the shared space that provide the actual computations of the distributed application. The MUDShell is the client of the MUDWeb application, and is the interface framework where the interaction with the user takes place: the MUDShell provides primitives for moving from one available shared space to another, and it allows the user to interact with the services by providing a MUD-like text box for direct commands, and by displaying the most common ones on additional buttons. Furthermore they allow avatars to display their interfaces just like the other services.

5.1 A conference management system

MUDWeb can be used not just to implement interactive games, but also for supporting a complex workflow. We have used it to design and implement a workflow system supporting the management of a scientific conference. Although there are probably more direct ways to implement workflow support for a conference, we found it interesting that such a seemingly remote metaphor, the MUDWeb, could be fruitfully used for this purpose. Our system, named *ConfManager*, has the goal to simplify the management of the review of papers submitted to a scientific conference.

Submitted papers are stored in rooms; authors, reviewers, and program committee members are represented by avatars which support both synchronous (online) and asynchronous (e-mail) communication interactions.

ConfManager includes the following rooms:

- `SubmittedPaper` Every paper is stored in a room of this type, that is dynamically created when the paper is submitted. The room will also store the reviews when they will be ready.
- `ReviewRoom` is used by reviewers to store their reviews before they are finalized.
- `SelectRoom` is a room accessible to the program committee members only; it stores the scores assigned to papers.
- `Papers` is a room reserved to the conference organizers: it contains managing data like the full list of submitted papers and the address data of authors.

The workflow related to the management of a conference is implemented by the interaction of the agents and the services. Each service activates whenever

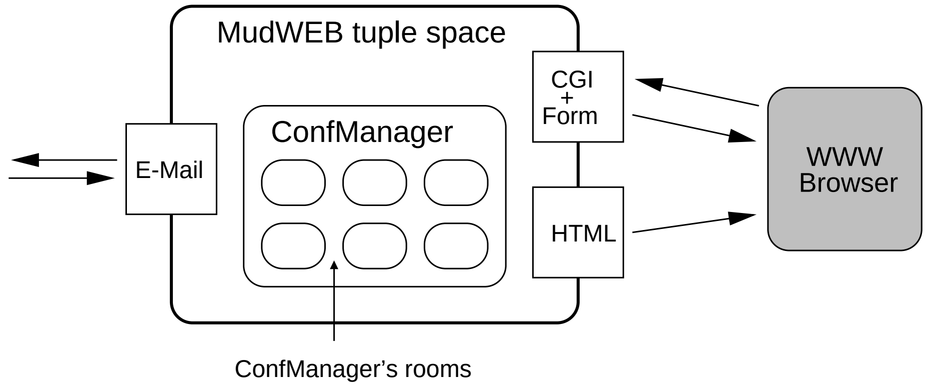

Fig. 3. Conference management mapped onto MudWeb

there are in the room it controls the items it needs to perform its activities. *ConfManager* offers in each room some specific services:

– **Services for authors of papers**: When a paper arrives it is stored in a room created on purpose, also including an avatar representing the corresponding author. The avatar can answer simple questions on the status of the submission.
– **Services for reviewers**: Each *SubmittedPaper* room, created to store a submitted paper, can also store reviews for the paper. User input is handled by some HTML forms and a few CGI scripts. The CGI scripts generate a review form and generate a tuple for each review. The tuple is stored in the same room containing the paper being reviewed until it is "confirmed"; after the confirmation the tuple is moved in the selectRoom.
– **Services for PC members**: PC members can act like both reviewers and authors. The service *Services.Selector* that can be activated in the *SelectRoom* room allows a PC member to choose what kind of interaction he is interested in.

All the tasks mentioned in section 2.2 are supported and available to the appropriate users. All of them are implemented by services that can either be activated by the users or by other agents.

All services come in two flavors, synchronous and asynchronous. For instance, the service *Services.Announcer* accepts reviews coming by e-mail, checking that they refer to the paper stored in the same room. The service *Services.Submitter* is similar to the preceding one, but supports on line user interaction.

Asynchronous services rely upon *avatars*, which have to be programmed to perform the necessary tasks. The (server-side) avatar acts as an e-mail client of an e-mail server; it controls the user mailbox and processes the messages it contains.

6 Conclusions and Future Work

We have presented a case study in document-centric groupware. 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 can be used to compare modern object oriented middleware infrastructures: for instance, we are developing a conference management system based on Lotus Notes. Another issue we are exploring is security, that is especially important for multiuser applications.

As we pointed out before MUD's robot players can be seen as synthetic users or as active items.

As of now 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 "active"; i.e. are subject to auto-modifications.

A document that represents a stock exchange's stock, for example, should periodically update its own value. A document that represent a contract should change its own state (and maybe it should also warn some users) when its expiration date is reached. We usually refer to this class of documents as *active documents*. In a MUD, the mapping of active documents into synthetic players is not natural and often not even 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 semantic.

XML [BPS97] is an extensible document markup language that provides a unified framework for describing orthogonally a document structure, its rendering, and its semantics. Introducing a technique that we call "displets" [CV99], our workgroup has integrated XML with Java, thus we can use a Turing-equivalent language for manipulating XML documents. The purpose of displets is to have active documents that are able to render themselves: we are planning to re-use the same concepts that are at the base of the displets to implement some more general-purpose kind of active documents.

The integration of this kind of active documents into the active items of a MUD environment based on MJada is very easy and opens new and interesting opportunities to design an active document management system based on the MUD metaphor, the PageSpace architecture and the MJada technology. We are currently engaged on this work.

Acknowledgments. PageSpace has been supported by the EU as ESPRIT Open LTR project #20179 as a joint research with people in TU Berlin. The case study requirements have been defined together by P. Ciancarini (Univ. of Bologna, Italy), R. Tolksdorf (TU Berlin, Germany) and O. Nierstrasz (Univ. of Berne, Switzerland), with partial support from EU WG "Coordina" project. The author wish to thank Paolo Ciancarini who has been the coordinator of the project of which this work is part of.

References

[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.

[CV99] P. Ciancarini and F. Vitali. Managing complex documents over the WWW: a case study for XML. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):(to appear), January 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.

[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.

[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 Barry Jacobs. Electronic Management of the Peer Review Process. *Computer Networks and ISDN Systems*, 28(7-11):1523, November 1996.

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

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

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