Using the Web to Coordinate Distributed Applications

P. Ciancarini
Dept. of Computer Science - Univ. of Bologna
Mura Anteo Zamboni 7
40127 Bologna - Italy
ciancarini@cs.unibo.it
http://www.cs.unibo.it/~cianca

R. Tolksdorf
Technische Universität, Fachbereich 13, Informatik
Sekr. FR 6–10, Franklinstr. 28/29
D-10587 Berlin, Germany
tolksdorf@cs.tu-berlin.de
http://www.cs.tu-berlin.de/~tolk

1 Introduction

Original Web middleware does not provide support for multiuser applications, such as groupware or workflow, as its basic nature is that of a passive information system. In fact, all activity is tied to server machines able to execute code using the CGI mechanism.

Enhanced web browsers supporting applet scripting languages, such as Java [1], allow multi-thread activity at the user interface. However, there is no integrated middleware to coordinate activities tied to multiple, distributed clients. Coordination has to be centralized at some server to which all users participating in an application have to connect to. Thereby, the activity located at the browser does not really make the application distributed, as applets at the browser cannot connect to other applets providing services to them directly.

For instance, think about a service to play chess or poker on the WWW against remote players. Java-enabled browsers can easily be used to build user interfaces including an interactive chessboard and other services, like a text-based messaging system. However, to coordinate a game over the Internet the Java programmer has to use sockets to transmit data and synchronizing events. This is a low-level approach subject to a variety of design and implementation errors.

In order to solve such a class of problems, and simplify the design and implementation of distributed applications over the Web, we present here the architecture of the *PageSpace*, a software platform based on extending Java with coordination technology [3].

The PageSpace introduces a notion of active, coordinable objects - the agents - which are able to both use and provide services from an to other agents, without requiring centralized coordination from servers.

Distributed applications involve heterogeneous machine-, network- and operating-system architectures. Coordinating agents in these environments means to make these heterogeneities transparent to the programmer. Using the PageSpace, transparency of network heterogeneity is achieved by using the Internet middleware underlying the communication platform. Transparency of different hardware architecture and operating systems is provided by Java. PageSpace is built on a set of existing technologies:

Coordination technology provides the conceptual platform for the coordination of activities amongst asynchronously working collaborating agents. Coordination technology has been initiated by the language Linda [3]. Linda is a minimalistic language,namely it includes a small number of constructs which need to be integrated in a conventional language to get a complete language for distributed programming. The basic conception of uncoupled coordination implicit in Linda has been studied in a large variety of research projects, focusing on parallel, distributed, and open distributed systems, on its theoretical foundations by giving semantics to coordination languages, and on a number of implementation oriented research concerning the embedding of coordination languages and their efficient implementation. For a sample of current researches, see [5].

Web technology provides a wide-spread communication and presentation platform to PageSpace. Browsers allow at least the access to the PageSpace and with HTML a uniform graphical interface for the applications can be defined.

Java technology provides a uniform processing platform. Java is popularized by its integration into standard Web-browsers and is available on all major hardware platforms.

The PageSpace integrates these basic building blocks and thereby adds value to them. PageSpace uses Java as the implementation language for the platform, as well as the application programming language enhanced with a high level coordination support for distributed, concurrent agents to Java. The PageSpace adds value to the underlying building blocks by integrating them in a manner in which the extensive reuse and combination of existing and wide-spread complementary technologies leads to a platform that aims at supporting real-life applications. Our guideline is to add the coordinating glue to bind these technologies together, rather than extending them individually.

2 The PageSpace Architecture

We identified the centralized nature of the current Web as its major drawback: all actual processing for the application is tied to the server-machine. Conversely, Java based applications are client-oriented. There is scarce integration between activities on the client and applications on the server.

Also, in most advanced distributed applications on the Internet, the WWW is only used as a static access mechanism to an existing external environment: eg. DCE components, CORBA objects, DBMS applications.

More important, we must not forget that designing and especially using distributed applications is difficult. Users, even sophisticated, have difficulties in understanding the intricacies of mutiuser distributed applications. This, it is necessary to explore new approaches to the design of such applications. We believe that the technology of coordination languages and models [3, 5] can help in simplifying our task.

The PageSpace architecture we are developing distributes the activities by incorporating coordination technology and decentralizes applications. Conceptually, a PageSpace is both a message repository and an environment for agents' activities.

2.1 The Agents of PageSpace

The PageSpace is populated by agents. Some of them are concerned with the application itself, others have more specific tasks. Fig.1 shows the overall abstract architecture of a PageSpace.

We distinguish the following entities in a PageSpace, each denoted by a different Greek letter:

- α An Alpha agent is an interface, by which users connect to the PageSpace. It is located inside a browser client which connects to a PageSpace. For instance, an Alpha could be a Java-enabled browser with one or more applets running, which connect to the PageSpace.

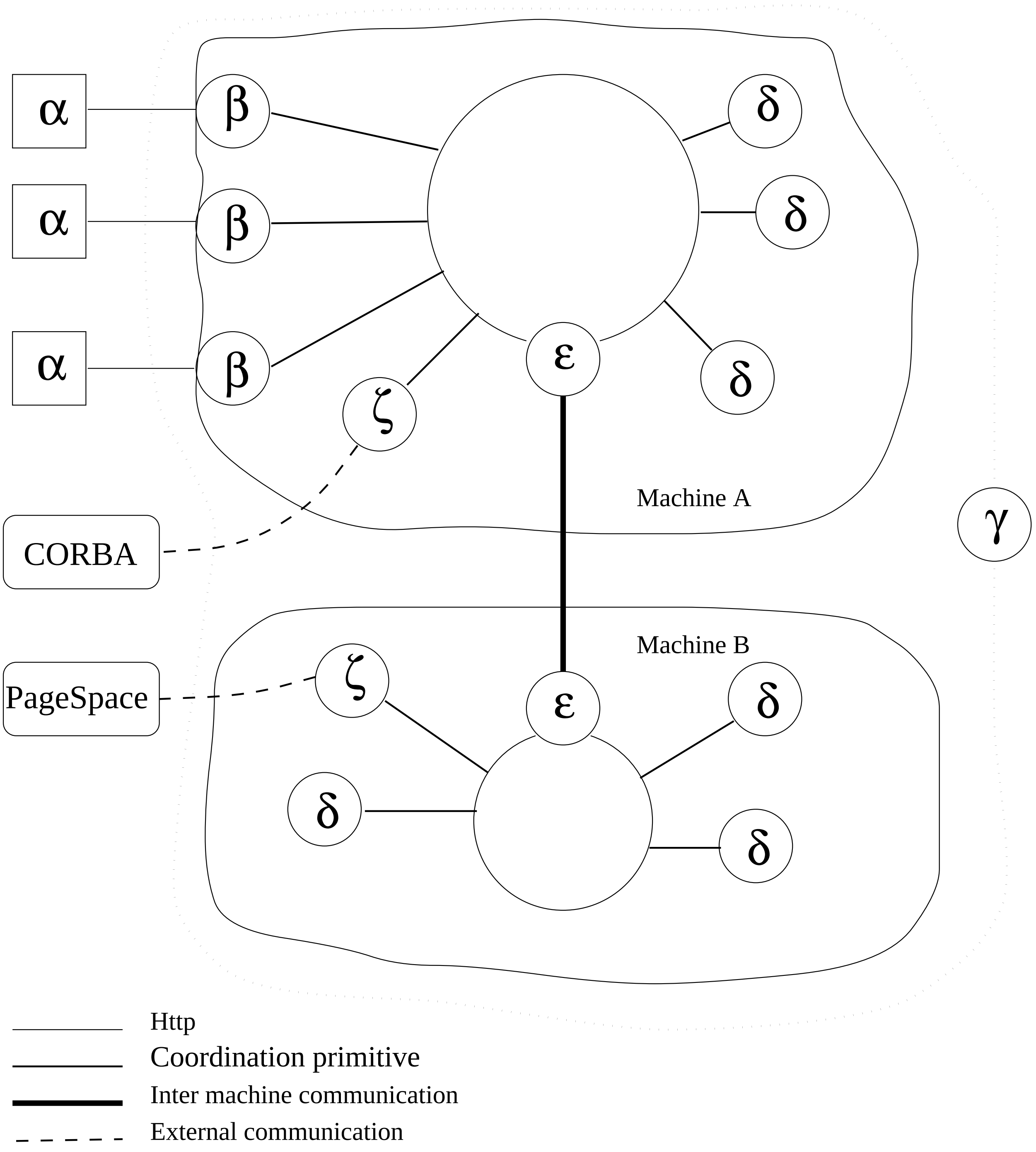

Figure 1: PageSpace Architecture

- β A Beta agent is the "homeagent", namely a broker which is the user representative within the PageSpace. It requests services on behalf of the user and it is independent from the user's Alpha. It is also persistent.

- γ Gamma is a coordination environment: it provides the glue which enables all agents to coordinate their work, either by Linda-like communication and synchronization primitives, or with more complex coordination protocols. For instance, we have made experiments with three different coordination models: Shade [4], Paradise [11], and Laura [12].

- δ A Delta is an agent which implements some application (or part of a distributed application). For instance, the coordination typical of a chess game (moves, time management, etc) would be implemented by specialized deltas. The Delta agents offer and ask services to other agents. They are active processes distributed somewhere in the PageSpace.

- ϵ An epsilon agent is actually the process implementing a Gamma environment on a workstation. Exactly one Epsilon agent resides on every machine in the PageSpace. It offers services to Deltas requiring knowledge about location and privileges, such as the ability to create a Delta. For instance, the JVM augmented with Linda-like primitives [6] could be seen as a specific epsilon for implementing some Gamma environment.

- ζ A Zeta is an interoperability broker, or gateway. The Zeta gateways integrate services provided within other coordination services, such as other PageSpaces, or distributed objects platforms like CORBA or DCE. Zetas are also the way to wrap legacy application in PageSpace.

We have chosen this way of presenting the PageSpace architecture because we needed neutral names to identify all components, which could be partially thought as organized in a stack of services. By "neutral" we intend for instance Java-independent: in fact, a number of alternative Internet languages are coming (eg. Inferno from Bell Labs), and we intend the PageSpace concept to be independent from any of them.

An Alpha interface connects via HTTP to the associated Beta broker agent. This carries out interactions within a Gamma coordination space with some application-specific Delta agents on behalf of the user. The Epsilons provide special services to the Deltas which are mainly location dependent. All agents use coordination primitives provided by the Gamma environment. A Zeta provides a gateway to other worlds, e.g. the CORBA world.

All agents (except possibly Alphas) are programmed in Java using special classes and executed as byte-code by Java virtual machines.

2.2 The Beta broker

We expect that the user will access the PageSpace with a WWW browser. Yet, there is no way for a WWW browser to be called by the WWW environment, since the HTTP protocol is directional and subject to the initiative of the client. This is considered positive since it allows human users to maintain control of the interaction and leave them autonomous and free of coming and going and doing other things at the same time. The role of Beta is thus that of a broker, namely a software agent which acts on behalf of a human user within the PageSpace, and which is the single access point to the shared environment.

2.3 The Delta Agents

We are supposing that several independent applications run within the same PageSpace. For instance, some players are playing poker, others are taking part to a distributed auction. Within the PageSpace, only agents interact - whose user interface is transported to Alpha. The autonomously interacting agents that are part of applications, are called Delta agents and reside constantly on the PageSpace, where they use and provide application specific services. For instance, there will exist a poker agent for every player, indirectly connected with the human user via Beta and Alpha and coordinating with the other agents in order to play the game. The software agent can be assumed to be reliably accessible by other agents for the whole length of the game, while the user can come and go at her leisure.

Deltas can exhibit a varying degree of autonomy for responding to other agents requests even in the absence of the human master. The response can range from a simple "My master cannot respond" to a more complex and autonomous behavior. For instance, a user can instruct her auction agent to wait until the object she is interested in is being offered and then perform offers of 100 dollars higher than the current level until the maximum price of 5000 dollars is reached, and then leave.

Communication among Delta agents (and Betas) uses the coordination technology in PageSpace. Thus, messages are exchanged by an I/O mechanism to a shared information space. The Delta-Delta protocol is application specific: A poker agent will speak a Poker-game-protocol, while a chess Delta speaks a chess protocol, each requesting or providing poker or chess services from others. In addition to these application specific messages, a basic set of common services is defined for Deltas, such as telling others about their location.

2.4 The Epsilon agents in a Gamma coordination space

Epsilons are agents which have administrative tasks. They are able to do privileged things that are prohibited to Delta and Betas. Each machine belonging to the same PageSpace has exactly one Epsilon agent constantly running.

Epsilon has two faces. On the one hand, it offers a set of common privileged services to Deltas and Betas (Start a Delta with some state, stop a Delta by brute force, and keep a log of started Deltas; store checkpoints of Delta-states somewhere; tell Deltas about their location and initiate the move of a Delta to some other machine).

In addition, Epsilon actually implements "behind the stages" the coordination structure which we call the Gamma environment. It is based on concepts from Linda. The role of Epsilon is central to our architecture. Its implementation goes far beyond just providing the common services. Epsilon is responsible to manage the Beta user brokers, just as a Web server manages homepages. In order to provide the access mechanism to Betas, Epsilon includes an HTTP server. Epsilon includes all mechanisms concerning the coordination platform.

2.5 Mobility of PageSpace Agents

The Delta-Delta protocol in the PageSpace is location independent, as other agents are addressed associatively, or by name. The Delta-Epsilon protocol is implicitly location dependent, as it addresses the single Epsilon agent co-located with the Delta agent. In order to move an agent, there must be some way to express the destination location of an agent. An agent should be able to ask for its current location. This can be achieved by a simple service of the Epsilon agents, allowing a Delta to communicate its location to other agents.

At least three situations may trigger the move of an agent to a different machine:

1. The agent itself has some reason to move. The agent could be able to perform a reflection about the effectiveness of its current location with respect to anticipated communication or computation demands. Also, an agent can make some observation about the state of its current location, for example to leave a heavily loaded machine.

2. Some other agent could have some reason to ask an agent to move. If, for example, an agent enters a game a sequence of further interactions with the other participating agents can be anticipated. The other agents could decide to ask the new agent to join them on some specific machine.

3. An agent with appropriate authority could force another agent to move. This could be due to some critical situation (heavy workload), or some policy to be applied ("This machine is mainly intended for weather forecasting. All poker-agents, please leave immediately!").

When an agent knows where to move to, it can initiate the move. A move comprises three issues: Removing the agent from its old location, transferring the agent, and starting the agent at its new location.

In our architecture, the move and transfer of state is handled by the Epsilon agents. A Delta agent uses a service of its Epsilon. Epsilon, in turn, invokes a remote Epsilon, which restarts the moved Delta agent an initializes its state. This protocol emphasizes the role of the Epsilon as the only agent able to address another agent at some specific location. Also, Epsilon is the only agent that has to have hold of the byte code of a Delta agent.

3 Conclusions

By combining three existing technologies - namely WWW, Java, and coordination-technology - it is possible to build a flexible platform to support open distributed applications. The necessary mechanisms turn out to be simple and require no extensions to the building blocks used. The protocols needed seem to be simple and scalable. PageSpace adds value to its building blocks by providing a highly abstract coordination mechanism for the development of open distributed applications.

The theme of building distributed applications on the WWW is gaining momentum. At the last WWW conference in Paris, several papers discussed a variety of approaches to accomplish such a goal [2, 8, 7, 13].

From our point of view the most interesting approaches are those which enhance the Web with coordination technology, like for instance WWWinda [9] and WULinda [10]. The former is a Web accessible interface to a local Linda tuple space; the latter uses local Linda-like coordination between a browser and its helper applications. The PageSpace approach is different in that it makes use of coordination technology to build distributed applications.

The PageSpace is currently under development; further information can be found at `http://www.cs.tu-berlin.de/~pagespc`.

PageSpace is supported by the EU as ESPRIT Open LTR Project 20179.

References

[1] K. Arnold and J. Gosling. *The Java Programming Language*. Addison-Wesley, 1996.

[2] M. Brown and M. Najork. Distributed Active Objects. *Computer Networs and ISDN Systems*, 28(7-11):1037–1052, 1996.

[3] N. Carriero and D. Gelernter. Coordination Languages and Their Significance. *Communications of the ACM*, 35(2):97–107, February 1992.

[4] S. Castellani, P. Ciancarini, and D. Rossi. The ShaPE of ShaDE: a coordination system. Technical Report UBLCS 96-5, Dipartimento di Scienze dell'Informazione, Università di Bologna, Italy, March 1996.

[5] P. Ciancarini and C. Hankin, editors. *1st Int. Conf. on Coordination Languages and Models (COORDINATION)*, volume 1061 of *Lecture Notes in Computer Science*, Cesena, Italy, April 1996. Springer-Verlag, Berlin.

[6] P. Ciancarini and D. Rossi. Jada: a coordination toolkit for Java. Technical Report UBLCS, Dipartimento di Scienze dell'Informazione, Università di Bologna, Italy, May 1996.

[7] S. Dossick and G. Kaiser. WWW Access to legacy client/server applications. *Computer Networs and ISDN Systems*, 28(7-11):931–940, 1996.

[8] N. Duan. Distributed Database Access in a Corporate Environment using Java. *Computer Networs and ISDN Systems*, 28(7-11):1149–1156, 1996.

[9] Y. Gutfreund, J. Nicol, R. Sasnett, and V. Phuah. WWWinda: An Orchestration Service for WWW Browsers and Accessories. In *Proc. 2nd Int. World Wide Web Conference*, Chicago, IL, December 1994.

[10] W. Schoenfeldinger. WWW Meets Linda. In *Proc. 4th Int. World Wide Web Conference*, pages 259–276, Boston, MA, December 1995.

[11] Scientific Computing Associates, Inc., New Haven, CT. *Paradise 4. Reference Manual*, 1996.

[12] R. Tolksdorf. Coordinating Services in Open Distributed Systems with Laura. In P. Ciancarini and C. Hankin, editors, *Proc. 1st Int. Conf. on Coordination Models and Languages*, volume 1061 of *Lecture Notes in Computer Science*, pages 386–402, Cesena, Italy, April 1996. Springer-Verlag, Berlin.

[13] J. Trevor, R. Bentley, and G. Wildgruber. Exorcising daemons: a modular and ligthweight approach to deploying applications on the Web. *Computer Networs and ISDN Systems*, 28(7-11):1053–1062, 1996.