Redesigning the Web: From Passive Pages to Coordinated Agents in PageSpaces

Paolo Ciancarini* Robert Tolksdorf† Fabio Vitali* Davide Rossi* Andreas Knoche†

Abstract

Currently, Web does not support distributed applications well. Existing approaches are oriented towards centralized applications at servers, or local programs within clients. To overcome this deficit, the PageSpace platform was designed for distributed, coordinated agents in the Web.

We take a specific approach to coordinate agents in PageSpace applications, namely variants of coordination language Linda that support rules and services to guide their cooperation. This technology is integrated with the standard Web technology and the language Java.

Several kinds of agents life in the PageSpace: User interface agents, personal homeagents, the agents that implement applications, and the kernel agents of the platform. Within the architecture it is possible to support fault-tolerance and mobile agents as well.

Keywords: Java, Linda, Coordination, Internet, Web Applications, Open Distributed Systems

1 Introduction

The Web has evolved into the dominating platform for information systems on the Internet. There is increasing demand to use it as a platform for distributed applications in which processing of information occurs. For example, the application domains groupware and workflow management require distributed access and processing due to the distributed nature of the work these applications support. Still there is no widely accepted platform for implementing distributed applications on the Web.

PageSpace is a platform that has the potential to provide sufficient functionalities to do so. It is based on the core Web technology for access and presentation, on Java as the execution mechanism, and on coordination technology to manage the interaction of agents in a distributed application. This paper describes the rationale for our platform, its design, and the implementation strategy currently applied.

This paper is organized as follows. In the next section, we review approaches to implement applications that require active processing on the Web. We then describe the technology, on which our specific approach to coordination in distributed applications is based. The next section describes the PageSpace platform and its agents. Then, the current approach taken in engineering and implementing PageSpace is outlined.

2 Existing Approaches for Web Applications

At its core, the Web is a static hypertext graph in which multimedia pages of information marked up in HTML are offered by servers, retrieved by clients with HTTP and displayed in a graphical interface that is very easy to use.

Because of its high availability, it becomes more and more desirable to use the Web as a platform for dynamic, distributed applications. The support of the core Web platform for applications is rudimentary – only the CGI mechanism allows for processing of information that is entered by the user in forms, or retrieved from auxiliary systems.

A number of mechanisms has been proposed and implemented to make the Web a platform for distributed applications. The following classification is structured according to the loci of activity possible with such mechanism.

- **Activity located at Web servers**. The CGI mechanism can be used to access other application servers from the Web. An example is database access, where a query is formulated in a form at the browser and a CGI script at the server passes that query – probably in a translated form – to some database server. The results of the query then are sent back to the users browser as HTML. Figure 1 shows that structure.

  However, with respect to the aspect of distribution of an application, this approach turns out to have nothing in common with distributed paradigms like client-server interaction, or others. In fact, interfacing an application via CGI to the Web does not mean to offer a distributed application. There is no processing at the

*Dept. of Computer Science, Univ. of Bologna, Pza. di Porta S. Donato, 5, I-40127 Bologna, Italy. *mailto:*{*cianca*|*vitali*|*rossi*} *@cs.unibo.it http://www.cs.unibo.it/*{*˜cianca*|*˜rossi*}

†Technische Universität Berlin, Fachbereich 13, Informatik, FLP/KIT, FR 6–10, Franklinstr. 28/29, D-10587 Berlin, Germany. *mailto:*{*tolk*|*knoche*} *@cs.tu-berlin.de http://www.cs.tu-berlin.de/˜tolk/*

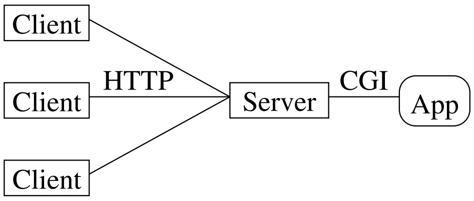

Figure 1. Application within one Web server

client besides displaying results. Moreover, there is only one central location of activity. Thus, such an application is basically a mainframe/terminal system at a large spatial scale. The Web server is like the mainframe – the only location of processing. The Web browsers are nothing but easy to use and graphical terminals, that use HTML as the display language.

- **Activity located at Web clients**. With execution mechanisms within Browsers, like the Java virtual machine, JavaScript, or plug-ins, activity can be performed at clients. If a Java applet is executed within the browser, it usually forms no distributed application. The applet is like a program that is run locally on the users machine. There is no generally accepted way to connect applets; the remote method invocation mechanisms lead to security problems that are not solved yet. If two or more applets communicate with some mechanism, they introduce an application specific protocol. Some applets and plug-ins connect to other proprietary servers and again leave the core Web. Figure 2 shows this structure of activity focused on clients.

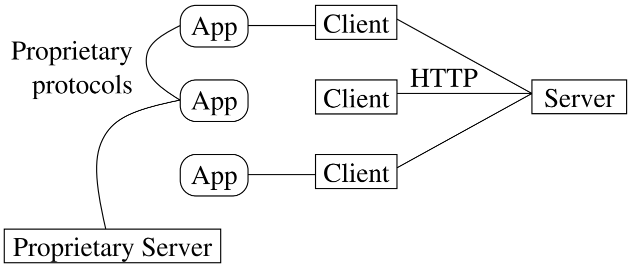

Figure 2. Activity at Web clients

- **Activity located in midware**. The previous two approaches are based on core Web technology, probably extended with proprietary communication schemes. A third approach to distributed applications on the Web is to use middleware to connect the active parts in an application, which can be located in clients and/or servers. Here, the Web technology takes the rôle of providing a uniform access and presentation mechanisms. Figure 3 depicts that structure.

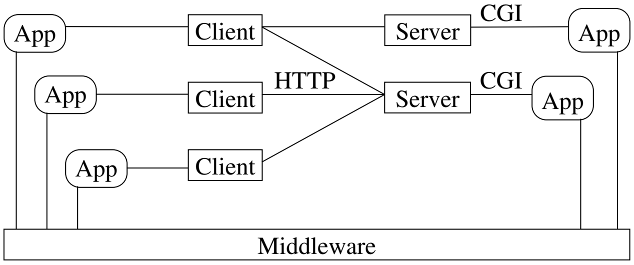

Figure 3. Activity within middleware

Examples for such approaches are [BIV+95], [HK94], or [AJ95], where the Web is an access mechanism to CORBA- or DCE-based applications, or integrated with middleware to profit from its services, such as secure communication. Client side middleware access is enabled by Java-CORBA embeddings like Sun's JOE.

The PageSpace platform falls into the last category, as only here activity is really distributed. We provide a middleware platform that is smoothly integrated with the dominating core Web technologies and also addresses issues of integrating the user interfaces of applications with the Web. The key conception is the use of coordination technology to manage the interaction amongst PageSpace agents.

3 Coordination Technology for the Web

The PageSpace platform ([CKTV96]) is based on the notion of agents that use coordination technology for their interactions. We use the term agent reflecting that processing is performed in such an entity. Applications are composed by a set of distributed agents. Each user has a *home-agent* that provides the interface to the PageSpace and its agents. We rely on Java as the main implementation language for our agents. In the main focus of the PageSpace is the issue of coordination amongst these distributed, concurrent agents, and we explored the use of Linda-like coordination technology to solve that coordination problem.

3.1 Basic Coordination Technology

Three issues are important in a distributed application: How do agents synchronize their work, how do they communicate, how is activity started? Amongst the various approaches to solve this coordination problem, is one line of research called *coordination technology* that is based on the concepts introduced by the language Linda ([CG89b]).

Linda introduces an abstraction for programming concurrent agents and defines a very small set of coordination operations. In a program based on Linda, a set of agents work on a task within a shared environment, called the *tuplespace*. It is a collection of tuples that contain information relevant for the application. Variants like distributed, or hierarchically structured ones, have been studied.

Linda's primitives provide means to manipulate that shared tuplespace, thereby introducing coordination operations. A tuple can be emitted to the tuplespace by an agent performing the `out`-primitive. As an example, `out(<" amount",10,a>)` emits a tuple with three fields, that contain a string, an integer, and the contents of the program variable `a`. This operation is non-blocking.

Two blocking primitives retrieve data from the tuplespace: `in` and `rd`. Both take a *template* as argument – for example `in(<"amount",?int,?b>)`. A *matching rule* governs the selection of a tuple from the tuplespace: The template and the tuple must be of same length, the types of the fields must be the same, and – for a constant field (an *actual*) – values of fields have to be identical.

The example pattern retrieves a tuple that contains the string `amount` as the first field, followed by an integer, followed by a value of the same type as the program variable `b`. The notion `?b` means that the retrieved value is to be bound to the variable `b` after retrieval. The difference between `in` and `rd` is that the former removes the matching tuple, while `rd` leaves it untouched in the tuplespace. Both operations are blocking as long there is no matching tuple found in the tuplespace. Linda makes no further guarantees on the selection of matching tuples and waiting operations.

It has been demonstrated ([CG89a]) that Linda is capable to express all major styles of coordination in parallel programs. `in` is a very powerful operation – it combines synchronization (the operation blocks until a matching tuple is found) with communication (the binding of values to program variables). Linda's operations together form a so-called *coordination language* ([GC92]). Combined with a sequential programming language, a new language for concurrent systems is generated. This combination is called *embedding* and can be implemented by changes to the programming language, by preprocessing source code, by libraries, or can be provided an extended operating system.

The following characteristics make Linda-like coordination attractive for distributed applications on the Web:

- **Uncoupling of agents**. The tuplespace as the coordination media uncouples agents in space and time. An agent can perform an `out` even when the retrieving agent does not yet exist, and can terminate before the `out`-ed tuple is retrieved. The tuplespace is a very high-level abstraction from locality issues.

- **Associative addressing** by using a template to retrieve a tuple means to state what kind of tuple is sought, not what tuple. This addressing is more abstract than retrieving a specific message.

- **Asynchrony and concurrency** are implicit notions of the tuplespace abstraction.

- **Separation of concerns**. The conception of coordination languages focuses on coordination only. It is not influenced by characteristics of the used programming language and leads to a clearer coordination model.

3.2 Coordination Technology in PageSpace

Coordination technology based on Linda uses repository of shared elements and operations for the addition and withdrawal as its core. To use this basic coordination mechanism for the Web, a Linda embedding into Java was defined for PageSpace and implemented. This system – derived from *Jada* ([Ros]) – forms our coordination kernel.

However, the pure data oriented style of coordination as in the original Linda-conception is not suited to support open distributed applications. It can well be used to keep state within one application, but it becomes difficult to support multiple applications that share one tuplespace.

With operations from two other coordination languages, we introduce two additional flavors of coordination styles:

- **Rule oriented coordination**

  The plain coordination mechanisms in Linda can be raised to a higher level by allowing declarative rules on coordination. ShaDe ([CCR96]) is an object-based coordination language. It offers a basic abstraction called the Object Space, that is similar to a tuple space with the difference that it contains objects. In fact, the Object Space is a distributed collection of objects and messages. Each object encapsulates a state in form of multiset of tuples and methods as rewriting rules.

  ShaDe objects are *active* units of computation with the ability to react to messages sent by other objects with an internal activity defined by methods. The state of an object is a multiset of tuples, so that the object itself can be considered as a tuple space, whereas the object space is a meta tuple space supporting inter object associative communication. Objects can use unicast, multicast, and broadcast communicate.

  For PageSpace, the most interesting feature of ShaDe is that coordination is expressed by rules. We intend to exploit such a feature to build "coordination" services enacting declarative cooperation laws. In the first prototype ShaDe was matched with Prolog, to obtain a distributed logic programming language. For PageSpace ShaDe is implemented on top of Jada.

- **Service oriented coordination**

  The notion of services is well adapted in open distributed systems. If can also form the basis for service oriented coordination languages that support the basic interactions in service-usage and provision.

  Laura ([Tol96]) is a coordination language designed for open distributed systems. Here, the tuplespace containing data is replaced with a *service space* containing forms describing service offers, requests, and results. The respective coordination primitives are `serve`, `service`, and `result`. Matching is performed on the service interface that is included in each of these kinds of forms. A subtype relation amongst these interfaces guides the matching routine in the selection of offers matching a service request.

  With a Laura reimplemented in Java, PageSpace agents are able to use and offer services at interfaces with Laura's coordination operations.

4 Kinds of Agents in PageSpace

In the PageSpace architecture, we distinguish several kinds of agents, denoted by Greek letters:

- **Alpha** agents are the interfaces of applications. They are manifested as a display in the users browser and can be "written" in HTML, Java, JavaScript etc. According to the capabilities of the users browser, there may be different instantiations of Alphas.

- **Beta** agents (also called *homeagents*) are the persistent representations of users in the PageSpace. They are responsible for the generation of Alphas upon the users request, for providing the user access to the agents and applications within the PageSpace, and for collecting incoming messages from other agents for the user.

- **Delta** agents offer and use services, interact by shared data, and implement applications within PageSpace. We distinguish application Deltas that also provide a user interface for interaction.

- **Zeta** agents are gateway agents to other coordination environments, or wrappers for legacy applications.

- **Epsilon** agents act as the kernel on each machine connected by the PageSpace platform. Epsilons manage the access to the PageSpace with an HTTP server, control Delta agents and offer coordination for agents.

Figure 4 shows an application in the PageSpace. There is an alpha in the users browser which is generated by a Beta. A set of Deltas implement the functionality of applications, and a Zeta provides access to a CORBA based coordination environment. The PageSpace environment Gamma space is established by a set of Epsilon agents on different nodes. In

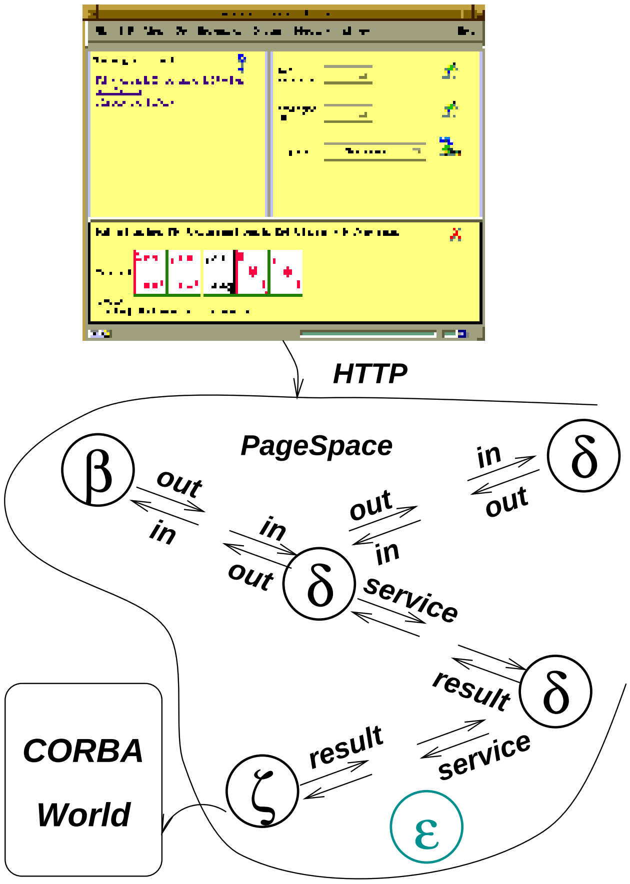

Figure 4. An application in PageSpace

the following, we describe the kinds of agents in detail.

4.1 Applications, GUIs, and Homeagents

PageSpace and its applications are accessible from any Web browser. This browser can be located at a different machine than the actual agent that performs an application. Also, the user can move during that interaction from one browser and machine to others. Thus, it is necessary, to deal with the user interface of an application separately.

As the interface has to be displayable by a Web browser, it is written in HTML. Due to the different characteristics of browsers, it can come in different formats. A text-based browser requires an interface without graphical components. Thus, we conceptually foresee that an application provides multiple representations of the interface.

The interface is moved from the application to the user, where the browser displays it and offers interactions. The processing of these interactions can take place at different

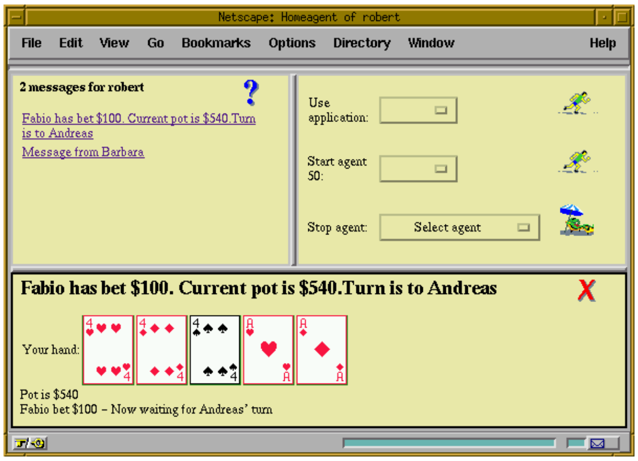

Figure 5. The user interface to PageSpace

locations – within the browser, if it is enabled by some mechanism like Java, or at the server, if it is form-oriented.

PageSpace has the potential to support any of the structures for applications on the Web as outlined in section 2:

- **Server located activity** can be supported by moving a static interface to the browser, which interacts with the application via a Beta agent located at a Web server.

- **Client located activity** by including a link to an applet within the HTML interface which does processing of inputs also. In PageSpace terminology, this means to transfer Delta agents to the browser.

- **Midware mediated activity** by transferring an applet with the interface that makes use of the coordination operations provided by PageSpace to interact with Delta agents distributed somewhere in PageSpace.

Each user of PageSpace has a persistent representation in the net, called the *homeagent* or "Beta". It has two faces – one to the user, one to the PageSpace. For the user, it provides the interface to PageSpace and the applications and agents therein. Figure 5 shows that interface for an example Poker application, which we call "Alpha", when it is manifested in the browser. This display is shown, when the user contacts his or her homeagent by retrieving a specific URL.

Alpha consists of multiple sections. One part of the user interface does provide operations of Beta – to use applications, and to start and stop own agents in the PageSpace. A list of messages shows the results of interactions with applications in the PageSpace. If the browser supports this feature, the queue is updated by a client-pull mechanism. When a message is selected, the user interface of that application is displayed in the third section of Alphas interface.

The other "face" of Beta is that it is a persistent representation of the user in the net. From Alpha, a user can use applications and start agents. However, he or she does not have to be online, while the application is running. Consider as an example a groupware application in which users all around the world participate in some work.

It is unacceptable to force users to be logged into the PageSpace all the time, as most distributed work is asynchronous in nature. Thus, while there is no connection to the user, Beta can still receive messages from a joined application. Beta looks like a complete agent to the PageSpace. However, it only stores incoming messages in a persistent store until the user retrieves them and reacts to them. In a future PageSpace, we will explore mechanisms to instruct Beta to automatically react to incoming messages.

4.2 Applications and Agents

Applications in the PageSpace are composed of agents, called "Deltas". There are three sorts of Deltas:

- An application Delta, which provides the specific user-interface of the application. As described in section 4.1, we separate the user interface and the application in order to access to it from Web browser. Still, some agent has to be contacted by Beta to get that interface. Any Delta returning such a GUI as the result of a method invocation is an application Delta.

- Deltas that implement the actual application specific functionality. A subset of a distributed application consists of agents that share an application specific context, which is of no use for other agents.

- Deltas used by an application, but which offer services to any application. These services are generic in that they can be used in the context of any application.

The agents that offer generic services can be started by a user within the PageSpace. They remain therein and answer to service requests by other agents until they are withdrawn.

The integration of legacy applications and gateways to other coordination environments can be achieved by wrapping and gateway agents that are called "Zeta". Like Deltas, they offer services to the PageSpace, but implement them by interacting with a closed application or via some middleware protocol to other middleware specific object.

5 Implementing the PageSpace Platform

The PageSpace is currently implemented as a prototype used for demonstration purposes and for experiments. This prototype follows the implementation strategy outlined in the following. Work remains to be done on the engineering of the platform, however, we believe that the main principles of our architecture can remain unchanged.

On each machine participating the PageSpace, one kernel Epsilon agent is running. Each Epsilon runs on a Java virtual machine and manages multiple threads. Figure 6 shows the logical outline of the Epsilon kernels. The several objects that run in threads are connected by streams for purposes of communication and management.

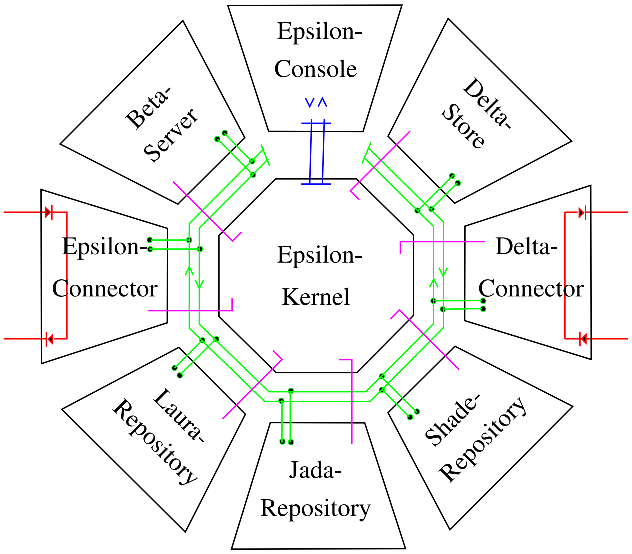

Figure 6. The logical structure of Epsilon

5.1 Provision of access to the PageSpace

Users access PageSpace via their homeagents that are contacted from a browser. Thus, a Web server has to be colocated with an Epsilon. As there are implementations of Web servers in Java – like Jigsaw from the W3C –, we integrate one of them as a thread in Epsilon. Thereby, interfacing of Beta to HTTP becomes much easier – instead of the CGI mechanism which only passes the CGI environment to a Java process, a call to a Beta object suffices.

5.2 Management of Beta agents

Betas are implemented by a single object within Epsilon. They are parameterized with the identification of a PageSpace user. After passing a login form in which a user name and password is entered, each user receives Alphas with the same components, but based on a different message queue.

The message queue in Beta is stored persistently in a database. Currently, we set up an mSQL ([Hug]) server for this purpose. Future databases written in Java, or interfaced with JDBC fit more smoothly in the implementation.

Besides the interaction with messages, the user can use applications, and start agents from Beta. Both result in the execution of a thread within the Beta object. That thread issues the appropriate coordination operation, waits for the results, stores it in the database and terminates.

5.3 Management of Delta agents

Each Delta agent is executed as a thread in Epsilon. This is reasonably, as we can make use of the native interaction mechanisms within one Java virtual machine for threads, and to avoid executing multiple virtual machines on nodes participating in PageSpace.

All Delta agents have the same kernel structure as depicted in figure 7. The main purpose of it is to pass invocations of the coordination primitives on to the Epsilon kernel. The specific coordination style can be supported by prefabricated handlers, for example for the dispatching of methods when the service-based Laura operations are used.

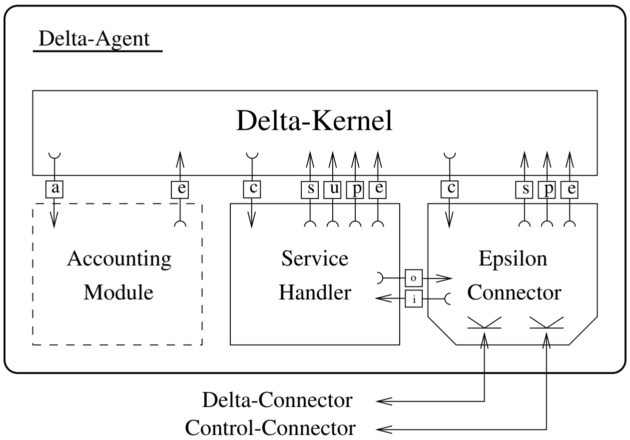

Figure 7. The logical structure of Deltas

Epsilon can easily manage its exception and monitor the operation of Delta threads. In a sense, Epsilon establishes the kernel of an operating system for Delta agents within a PageSpace coordinated Java virtual machine. We inherit thread management from Java, and add "process" interaction mechanisms by coordination technology.

5.4 Implementing the coordination operations

As outlined in section 3.2, we include several flavors of coordination technology in PageSpace. All of these have in common that they are centered around the use of a shared space of element of some kind, and that a matching rule guides the coordination primitives.

Thus, Epsilon contains instances of a generic basic component, the *repository*. These are collections of elements of some type. Each such repository implements the specific operations of a coordination language with a specific matching routine, thus is may be optimized, but still is based on the management of a pool of elements of some type. Within the Epsilon architecture, multiple repositories can be integrated to the internal control and data streams.

5.5 Identifying agents

Accessing services and naming are central issues in middleware. In PageSpace, we do not enforce a registry of agents, but take a different approach similar to how pages are accessed in the Web. As Epsilon knows about all agents it is managing, it is able to provide lists of these and their interfaces. The natural way to access such a list is by the built-in HTTP server. Thus, the "name" of an agent for the outside world is a simple URL.

Users that want to use agents, keep their personal list of known agents – just as one does for known Web pages with a bookmark-list. This list is used by Beta to offer the use of agents and applications to the user. It can be extended by the user, and be the subject to public catalogs of agents – resembling search engines and index services on the Web.

For a service-based use of PageSpace, the interface of a requested agent has to be passed to the coordination operations. This interface is accessible from the Epsilon that manages an respective agent. Beta retrieves it via HTTP, and constructs the appropriate coordination operation.

5.6 Distributed PageSpaces

The Epsilon kernels manage and coordinate agents on one machine. For distributed applications, these kernels have to have a distribution architecture and according protocols. A special concern with such a protocol is scalability – the ability to provide efficient coordination for a platform involving a large number of machines.

Establishing a shared repository of information can lead to scalability problems due to the amount of overhead for replication. We can take a flexible approach to structuring the system to overcome these problems. We follow the approach of the Internet to scalability: The set of machines that participate in the PageSpace is organized in a loose hierarchical fashion: Locally connected machines follow a replication schema in a logical sub-PageSpace and one machine is defined as the gateway to other sub-PageSpaces. Thereby, we imitate interconnected LAN the Internet.

The specific organization of Epsilons within one sub-PageSpace is a local decision. Known architectures for distributed implementation of Linda-like systems include full replication of a repository to all nodes, no replication with a centralized repository, or a partial replication as in [CG86]. As long as there is one defined node that runs a gateway protocol to other sub-PageSpaces, our architecture supports all of them. In fact, the current Jada implementation uses a centralized or fully replicated repository, whereas Laura implements a partial replication scheme.

For a gateway, a "routing-table" exists that instructs the gateway to which other sub-PageSpaces requests for matching elements shall be forwarded. Thus, the distribution structure can be statically or dynamically configured. This configuration will be based on the structure and behavior of the agents within a sub-PageSpace, and supports them in their coordination requirements. The several flavors of coordination employed in PageSpace give way to several intelligent optimizations, that are to be evaluated.

6 Features of the PageSpace Platform

The PageSpace architecture has several features that are yet to be explored. In this section, we point to two of them, namely fault tolerance and mobility. We show how these features can be introduced to the platform, and how they are enable by the design of the platform.

6.1 Fault Tolerance

The architecture of PageSpace opens perspectives to satisfy the needs for fault tolerance. Failures of the Alpha agents – because of a crashing browser, or a fault in the users machine – do not affect the PageSpace at all. The failure of a Beta agent does not introduce problems, as the queue of messages for a user is kept persistent.

The Beta, Delta and Zeta agents are managed by Epsilon. Thus Epsilon can keep a log of their external interactions and to request state information from that is stored persistently. Epsilon thus can monitor the managed agents, and restart them in case they crashed with a given state. We foresee that any of the managed agents can provide a method that transfers state information to Epsilon. The log of the external interaction can be used to keep the repositories within Epsilon fault-tolerant. The state information can be used to keep the managed agents tolerant to failures.

In the case of an Epsilon failure, the kernel and all managed objects are lost. The log of external interactions can be used to reestablish the repositories after restart; the managed agents can be restarted accordingly. An alternative would be to make the repositories themselves persistent, and the coordination operations transactional – however, the overhead involved has to be evaluated.

6.2 Mobile PageSpace Agents

As stated above, Delta agents interact location transparent. This fact, and the technical characteristics of the Java platform makes them candidates to establish a notion of mobility of agents within PageSpace.

To do so, agents have to pass their internal state to an Epsilon, and Epsilon has to be able to start an agent with a specific state. The use of agents compiled into code for the Java virtual machine together with its run-time-linking capabilities makes the code of the agents portable within the PageSpace environment.

Deltas may want to be moved because they detect that they interact with each other and try to make the coordination more efficient by "meeting" at a specific location. They can be asked to move by an authoritative Epsilon, because a specific policy applies to their current location.

In any case, the notion of location has to be introduced. We foresee, that a Delta can ask its Epsilon about the current location and that it is able to communicate it to another Delta. We do not foresee any operations on location representations available to Deltas. Epsilons can stop agents, transfer them to another Epsilon, which in turn restarts them. The state of Delta has to be passed along with the byte code of Delta. The access to that state is provided by Deltas, as foreseen in the fault tolerance mechanisms.

It has to be evaluated, what protocols are most efficient to perform such operations, and what strategies for mobility should be followed by Deltas and Epsilons.

7 Conclusion

The PageSpace is a platform to support distributed applications on top of the Web. We provide a framework that is based on the core Web technologies and Java, and add a specific approach to coordinating distributed agents in applications, namely Linda-like coordination technology. Our approach is generic towards the usage of several variants of coordination technology, as demonstrated with data-oriented, service-based, and rule-driven coordination styles. The design of the platform is enabling for a straightforward implementation of several desirable features, such as fault tolerance and mobility of agents.

The first phase of project PageSpace was concerned with the development of our approach, a prototypical implementation, and a demonstration of its potential. Now the focus is on engineering the platform, and on validating our conception with applications in the field of electronic commerce. Information on PageSpace can be found on the Web at `http://www.cs.tu-berlin.de/~pagespc`.

Acknowledgments. PageSpace has been supported by the EU as ESPRIT Open LTR project #20179.

References

[AJ95] G. Almási and V. Jagannathan. Integrating the WWW and CORBA-based Environments. In *Proceedings of the Fourth World Wide Web Conference*, 1995. (Web* Home Page: http://webstar.cerc.wvu.edu/lpi/).

[BIV+95] Ashley Beitz, Renato Iannella, Andreas Vogel, Zhonghua Yang, and Tak Woo. Integrating WWW and Middleware. In RS Debreceny and AE Ellis, editors, *Innovation and Diversity - The World Wide Web in Australia. AusWeb95 - Proceedings of the First Australian World Wide Web Conference*, 1995.

[CCR96] S. Castellani, P. Ciancarini, and D. Rossi. The ShaPE of ShaDe: a coordination system. Technical Report UBLCS-96-5, Department of Computer Science, University of Bologna, 1996.

[CG86] Nicholas Carriero and David Gelernter. The S/Net's Linda Kernel. *ACM Transactions on Computer Systems*, 4(2):110–129, 1986.

[CG89a] Nicholas Carriero and David Gelernter. How to Write Parallel Programs: A Guide to the Perplexed. *ACM Computing Surveys*, 21(3):323–357, 1989.

[CG89b] Nicholas Carriero and David Gelernter. Linda in Context. *Communications of the ACM*, 32(4):444–458, 1989.

[CKTV96] Paolo Ciancarini, Andreas Knoche, Robert Tolksdorf, and Fabio Vitali. PageSpace: An Architecture to Coordinate Distributed Applications on the Web. *Computer Networks and ISDN Systems*, 28(7–11):941–952, 1996. Proceedings of the Fifth International World Wide Web Conference.

[GC92] David Gelernter and Nicholas Carriero. Coordination Languages and their Significance. *Communications of the ACM*, 35(2):97–107, 1992.

[HK94] Edwin E. Hastings and Dilip H. Kumar. Providing Customers Information Using the WEB and CORBA. In *Proceedings of the Second World Wide Web Conference '94: Mosaic and the Web*, 1994.

[Hug] David J. Hughes. Mini SQL – A Lightweight Database Engine.

[Ros] Davide Rossi. Jada: multiple tuple spaces for Java á la Linda.
http://www.cs.unibo.it/~rossi/jada/.

[Tol96] Robert Tolksdorf. Coordinating Services in Open Distributed Systems with Laura. In Paolo Ciancarini and Chris Hankin, editors, *Coordination Languages and Models, Proceedings of Coordination '96*, LNCS 1061, pages 386–402. Springer, 1996.