Distributed Web Application Development with Active Web Objects

Gustaf Neumann
Department of Information Systems
Vienna University of Economics, Austria

Uwe Zdun
Specification of Software Systems
University of Essen, Germany

Abstract

Modern distributed web applications should offer high customizability, various communication resources, flexible data and document representations, persistence, metadata, mechanisms for interaction and coordination, etc. Often these requirements are either badly supported by the development tools or a diverse set of technologies has to be used which is orthogonal to web technology and is based on overlapping concepts, abstractions, and paradigms. In this paper we present ACTIWEB ***as a single framework centers around the notion of active web objects. Those integrate web documents with objects of an object-oriented scripting language. The scripting language enables rapid application development and component gluing. Moreover, several basic services, such support for XML, RDF, remote procedure calls, code mobility, object persistence, and object registration are provided.***

Keywords: web objects, distributed web applications, scripting

1 Introduction

The World Wide Web (WWW) was developed with a set of desired qualities, such as portability, interoperability, a simple form of scalability, and extensibility. These goals led to a simple architecture that is centered around document structures, but which also provides resources to develop web-based applications in a distributed client/server environment. Advantages, like the simplicity of the architecture, the ability of human beings to understand the presented information directly, the ease of adding new information, and the ease of connecting information pieces through links have led to a dominating position for web-based information systems. But this simple architecture has several drawbacks for distributed web application development:

- It lacks support for interactive or collaborative multi-user applications [1] and is not well suited to exploit the benefits of today's distributed (object) technologies.
- The basic data structure of the web – hyperlinked HTML pages – is too restricted to support complex applications [2]. The integration of important services, such as diverse communication infrastructures, data representations, metadata, persistence, interaction and coordination of web-based applications, etc. is often missing.
- Current web object models, as in [3], lack integration with these services and do not provide powerful abstractions of modern OO languages.
- Web-application development in system languages, such as C, C++, or Java, e.g. with the CGI interface is often complex and misses the flexibility and customizability required by many web applications (see [4, 5]). To meet the requirements of developing and maintaining a dynamic web site, a developer should use tools and technologies that maximize flexibility and minimize development time.
- Building flexible, distributed web applications on top of current web standards is not impossible. However, distributed technologies/services and web standards, like the HTTP protocol, are often not well integrated [6].

In this paper, we present an extensible, component-based framework for distributed web-based applications that centers around the notion of active web objects. The basic concept is not new, it has been used in the context of other domain. However, the framework presented is implemented with today's web technology without suffering from the complexity of middleware approaches. It also avoids the limitations of CGI-like architectures, like problems with client-to-client interaction, customizability, performance, etc.

2 A Framework for Active Web Objects

This section presents the basic architecture of our ActiWeb framework. In this section, give firstly a very brief overview of the object-oriented scripting language XOTcl in which ActiWeb is implemented. XOTcl provides a set of uncommon properties that ease the implementation of our component framework. Then we discuss the conceptual hierarchy of the basic web object types, the integration with code mobility, and the base-line architecture of service components on top of a flexible HTTP implementation. In Section 3 we discuss the service components in more detail.

2.1 Extended Object Tcl (XOTcl)

Extended Object Tcl (XOTcl) [7] (pronounced *exotickle*) is an object-oriented extension of the language Tcl. Scripting languages gain flexibility through language support for dynamic extensibility, read/write introspection, and automatic type conversion. Tcl and similar scripting languages are designed as two-level languages, consisting of components written in efficient and statically typed languages, like C or C++, and of scripts for component glueing. The primary purpose of the scripted layer is flexible composition of components.

XOTcl enhances the "glueing of components" idea of Tcl with language constructs to support architectural fragments, such as design pattern parts, and to provide explicit support for composition and decomposition. All object-oriented constructs are fully introspectable and all relationships are dynamically changeable. XOTcl offers a set of basic constructs which are singular objects, classes, meta-classes, nested classes, and language support for dynamic aggregation structures. Furthermore, it offers the message interception techniques *per-object mixin*, *per-class mixin*, and *filter* to support changes, adaptations, and decorations of message calls.

2.2 Active Web Objects

The goal of active web objects is to provide a unifying entity for application development and for information exchange in a machine understandable manner. Web objects provide activeness for web artifacts. That is, web artifacts are represented by an object that contains methods defining the web artifact's behavior. In this context, the traditional web with interconnected web pages can be interpreted as a drastically simplified system of web objects, where the web pages can be seen as objects (unambiguously identified by URLs), and links can be seen as methods (invoked via a web server, which functions as dispatcher). We generally see each object that is accessible via a URL as a web object.

The notion of web objects also manifests in variants of dynamic HTML and the Document Object Model (DOM) [8] which treat HTML documents like programmable objects. In such approaches, the programmer can interact with document elements without parsing the text. However, DOM based approaches only offer limited suitability for general application development, since they are conceptually centered around classical web documents rather than providing resources for expressing the application semantics of distributed systems.

Therefore, our notion of active web objects goes further. An active web object is a full-fledged programming language object which is capable to represent itself on the web. An active web object contains methods and data, has a runtime state, a location, and an URL to be accessed via the web. Requests for the objects via the web (e.g. HTTP requests) are mapped directly to method invocations of the active web objects. For a client of the system, like a web browser, active web objects look exactly like traditional web pages.

An important property of a web object is its location. An active web object "lives" in the runtime environment of an HTTP server. The HTTP server has the responsibility to translate URL requests to method invocations on the object. In ActiWeb active web objects can also be called via direct RPC calls (on top of HTTP) without HTML markup.

In ActiWeb every object that can be accessed through a place is called a *web object*. Each *place* is uniquely identified by a host name and the port number of the HTTP server which provides access to the place. A web object is unambiguously identified through a URL comprising of host name, port number, and object name. There are different special web object types: A *web document* is a web object which contains a web artifact, such as an HTML page, a picture, a sound file, etc. An mobile *agent* is a web object that can exploit mobile code primitives, like *migrate* or *clone*. A *web agent* is a mobile agent that supports – beside the mobile agent abilities – views on the agent's state and behavior in various web representations, like HTML, XML, etc.

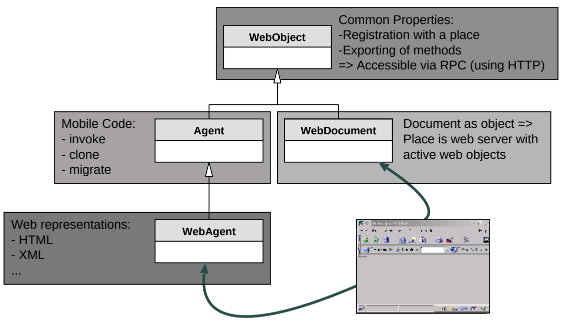

Figure 1: Conceptual Class Structure of Web Objects

Figure 1 presents the conceptual class structure of web objects. We call it a conceptual class hierarchy because in XOTCL a class hierarchy is highly flexibly and does not necessarily express only the intrinsic properties of an object. In XOTCL it is, for instance, possible to attach – dynamically and transparently – a class to an arbitrary object by per-object mixins [9] as an implementational role. Therefore in ACTIWEB an agent object can easily behave exactly like a web document and vice versa.

ACTIWEB is an object-oriented mobile code environment which integrates agent-technology and object-oriented programming with the current web infrastructure. This reflects our baseline architecture and terminology. To implement HTTP-based mobile agents, we require a central access point in the HTTP server. We also require such a central access point for integrating web documents with an object system. Therefore, the *place* abstraction of the mobile code paradigm is a good access point for the active web object system. It also serves for other central issues, like the mapping of URLs to methods of web objects or to provide security and access control issues.

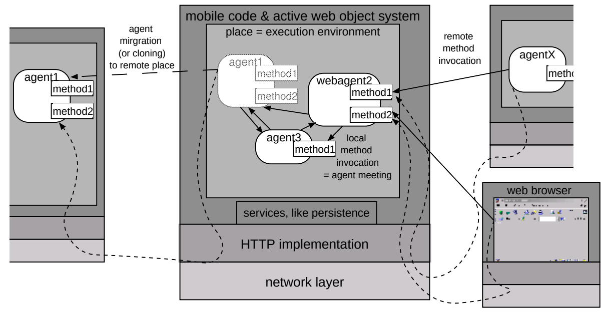

Figure 2: Mobile Code and Active Web Object System – Communication Resources

Figure 2 presents a snapshot with migration paths of an hypothetical application. Places form safe execution environments for active web objects (and other objects of the application). Each agent can communicate via RPC calls and clone/migrate. An web agent additionally contains methods to return a certain representation, like HTML or other web representations. The place maps URLs to method implementations.

3 Components of ACTIWEB

The ACTIWEB system (see Figure 3) is implemented in XOTCL. On top of the XOTCL layer a set of basic services is implemented. Generally all components can be substituted by compatible ones. XOCOMM [10] provides an object-oriented implementation of an HTTP server and HTTP access. All communication in ACTIWEB relies on this service. Places, the basic execution environments of ACTIWEB, contain exactly one HTTP server identified by host and port. ACTIWEB objects (like agents) have access to other (remote) ACTIWEB objects via HTTP.

The metadata service provides an object-oriented implementation of an XML- (called XOXML) and a RDF-parser/-interpreter (called XORDF). RDF metadata is used in ACTIWEB as a canonical data representation. XOSTORE is a general persistence service for XOTCL providing persistent objects. A registry service XOREG enables registration of ACTIWEB objects, say, to find an object through the specification of certain properties. XOMOS implements the mobile object system based on the lower level services.

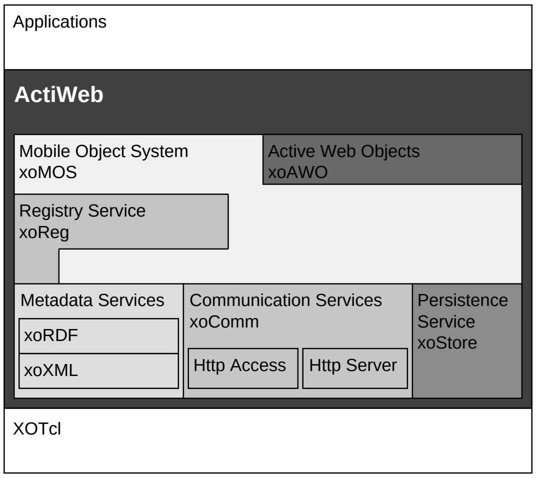

Figure 3: ActiWeb: Basic Architecture

The active web objects component xoAWO provides web-representations for active objects and agents. All components can be loaded on demand. The following sections describe these components in more detail.

3.1 Flexible Data Representation and Metadata Services

For flexible data representation an XML and RDF parser/interpreter framework is integrated in ActiWeb [11]. XML is primarily used as a flexible data glue and a platform independent data representation.

The problem handled by the metadata service is that data on the web is machine readable, but hardly machine understandable. The Resource Description Framework [12] is a formally defined model for description of web resources with metadata. RDF metadata can be expressed in various forms. The RDF data model itself is visualized by directed graphs, with two kinds of nodes for web resources and properties. But RDF metadata can also be linearized to XML.

In ActiWeb we use RDF metadata as a general form of knowledge representation about web objects. These include ordinary web documents which are described by metadata, like author, title, etc. But it also includes agents which have similar metadata properties and additional properties solely defined for agents, like the information an agent uses for migration. These are the code of the agent, the current state, and a start command with which the agent resumes its actions at the migration target. Agents migrate by automatically generating these RDF metadata and by sending them in an XML linearization.

3.2 HTTP-Based Communication Service

xoComm [10] is a communication infrastructure for web applications, based on the HTTP protocol. It provides an HTTP server and client access. Furthermore it is the basic communication service for the ActiWeb web object and mobile code system. The HTTP server component of xoComm is used to implement ActiWeb places. The places use the HTTP client access to provide the RPC communication means for their agents.

In ActiWeb each place is a Singleton [13] in one process at a distinct port. Each place aggregates a web server object. Per default all place communication is handled by the place's web server. Each ActiWeb agent invokes, clones, and migrates itself via HTTP access. Agents generally exploit asynchronous communication. In order to be able to exploit communication facilities agents must register themselves with a place. Therefore, from the point of view of a client agents act as HTTP clients and servers. The place controls which methods of which active web objects are accessible from the outside. All other URL requests are not redirected to the object, but produce an HTTP error. Furthermore, on the accessible objects we may add basic and digest access control.

3.3 Persistence Service

To let agents migrate to unknown hosts or even let them act autonomously for a while, the underlying mobile code system has to secure the agent for faults in foreign hosts. A persistence service enables recovery of agent from nonvolatile storage when a place has a fault or is turned down temporarily. But object persistence is also required in most distributed web applications which do not rely on code mobility. If an object persistence service is missing, it usually has to be programmed by hand, e.g. on top of a relational database.

The xoStore persistence service realizes persistence through an abstract storage interface that allows us to use different storage STRATEGIES with a unique interface. To let the storages be exchanged dynamically and transparently we attach them as PER-OBJECT STRATEGIES which are implemented conveniently using per-object mixins [9]. Legacy components – like databases – are attached to the special storage strategies using the WRAPPER FACADE [14] pattern. A persistence MANAGER allows us to use different storage suppliers at once and even exchange them at runtime to the most convenient storage form.

Agents (and all other objects that need persistence) may add persistence transparently and dynamically through PER-OBJECT STRATEGIES for persistence. Currently clients can choose between an eager strategy that writes changes in the object's data to the storage as they occur, and a lazy strategy that writes the object's data when the process terminates.

3.4 Mobile Web Object System

The components of the mobile object system implement an RP and RPC environment for mobile agents. Important components are mobile agents, places, and agent management. To let a web object be able to be called via a remote call, the web object class has a method `exportProcs` that lets an object dynamically specify which methods are currently exported for remote calls. Only these method calls are dispatched by the place. All other calls result in an HTTP error.

Every remote call is handled via the place. Places are unique to a process and unambiguously identified by host name and port. The place is a specializable SINGLETON [13]. All web objects are also identified unambiguously by an URL. To let object-oriented calls be invoked using URLs, we automatically transform them with the web standard CGI encoding/decoding (e.g., spaces are transformed to '+'). The general form for object-oriented calls via a URL is:

```
http://hostname:port/objName+methodName+arguments
```

An agent management component implements a special agent that fulfills the management tasks for agents of the place. All agents of a place have to register/deregister with the agent manager. It (lazily) creates RDF metadata on the agents code if an agent clones or migrates to a foreign host. Agent management also includes immigration from a foreign host. For immigration, RDF metadata that contains the agents code and data has to be transformed into XOTCL code. Then the start command that represents the last state of the agent at the origin host has to be evaluated.

The agent component extends web objects by allowing an agent to `invoke` RPC calls and by implementing RP abilities using `clone` and `migrate`. To let a place distinguish RPC and RP call, we use the HTTP method GET to denote an RPC call and the HTTP method PUT for RP calls. `invoke` takes an object-oriented call, codes it with CGI encoding, and asynchronously sends it via HTTP GET. `clone` calls the agent manager to lazily create an RDF metadata script, if it is not already existing. The script captures the current code and state of the agent. A dynamically processed start command is used to specify where the agent resumes its work at the foreign host. Finally, the agents is sent via a PUT request. `migrate` clones the agents and destroys it afterwards locally.

3.5 Registry Service

Sometimes an web object's services have to be searched in the network. Often object/agent interaction has to be coordinated. Such tasks can be solved by a service-based registry architecture: Each place contains a registry which is able to store properties for its objects. Objects can register themselves with a set of RDF metadata, like type, name, attributes, etc.

Other agents can send a request for properties to the registry agent. The request is compared to the attributes and a list of matching agents is returned. Another variant is to directly redirect the call to one of the matching agents via HTTP REDIRECT. A special service for agents that are not permanently connected can test from time to time whether an agent is still connected and removes it from the registry if not. Several registry agents in different places may be connected and can forward queries to other registries, e.g., in a hierarchical fashion (similar to the domain name service (DNS)).

3.6 Web Representations for Active Web Objects

Different users have different requirements for representations. Mobile agents allow us to give applications interfaces that can be invoked by other agents or that can migrate as mobile code to a user's host. In both cases an user interface that runs on the user's host has to be shipped with the user's client. Often it is hard to predict all required user interfaces, or it may not be possible to implement an interface for the user's platform. Active web objects, as in the ActiWeb system, are an extensible form of representations (including a web representation). A user can access one and the same object via several different representations.

An active web object can provide web representations that are accompanied or generated from the object's methods. There are two different kinds of active web objects: Special agents that also have a web representation and ordinary web documents, like HTML pages, pictures, etc. that also have active parts in form of methods. The two forms may also be combined to active web agents that contain a web document (e.g., if a document should migrate with an agent to a foreign host).

Through the extensible invocation interface of the place we add a representation invoker for every additional representation. A client accesses these representations through special Facade objects that hides the representation. The place acts as a Proxy and forwards calls for a special representation to the proper Facade. For clients the Facades hide the underlying subsystem's implementation through their representation.

Web documents are web objects that have the capability to attach or detach files, like HTML pages, pictures, etc. A document Factory allows us to automatically create objects with names comprising a directory and file name. Therefore, a whole tree of a web server can be automatically transformed into an object-oriented representation using MIME type guessing. For a client the ActiWeb system then acts as a normal web server for clients, but all documents may be extended by active parts. The object names (which are encoded in the URL) are identical to the filename part in the URL. E.g., the place can easily distinguish requests from different hosts and can behave differently for certain files. E.g., confidential files can be protected by access control, if they are accessed from outside the local area network.

4 Related Work

A solution of the preceding example using CGI interface would have several drawbacks in comparison to the presented solution. The architecture would be a loose coupling of scripts without a component concept which are hard to maintain. Most tasks would have to be programmed by hand instead of exploration of a service framework. CGI scripts do not support suitable direct client-to-client interaction. The CGI concept is not equipped with mobile code abilities. Therefore, it can suffer from problems of performance and customizability.

If we compare the example solution to a similar solution using a distributed object system, like CORBA [15] or DCOM, we can assess that both solutions are able to hide the networking details. Both offer an integrated component concept. For data collection and synchronization the mobile code solution may offer significantly better performance and customization abilities. The distributed object system does not have a solution for static and active web representations of an implementation and it does not provide various facades for one business logic. Several services, like registry, persistence, etc. have to be programmed by hand. Some of these approaches, like CORBA 3.0 or Java RMI/EJB, also offer a component model integrated with distributed objects.

The usage of a conventional application server (like WebLogic, WebSphere, etc.) offers a concept for giving a business logic an additional web representation. But it does not ease the usage of other representations. Furthermore the pure server-side approach means a significant network load for client to client interaction, as needed in the presented example for the project management implementation, the clients, the data collection and the synchronization.

Nevertheless, commercial products, like distributed object system and application servers, offer a great number of services, supported platforms, etc. that are still missing in ActiWeb. But through XOTcl's extensibility with C components commercial products written in C or

C++, like CORBA ORBs, transaction monitors, message queueing systems, etc. can relatively easy be integrated. Furthermore, ActiWeb makes use of several web standards, like the HTTP protocol, URLs, CGI de-/encoding, RDF metadata, etc. These are supported by most languages on most platforms.

Telescript [16] is an object-oriented programming language that pioneers in the area of mobile code. The terminology of this work is related to Telescript and several abilities of Telescript, like persistence, mobile code, etc. are implemented in ActiWeb. Telescript does not support web representations and is not integrated with web standards. D'Agent [17] is another mobile code system that supports the languages Java, Scheme, and TCL. Agents can clone, migrate, and send code to another agent. The system lacks the additional service, especially the web representation abilities. In [18] the general idea to provide one general framework for development of distributed and information-oriented applications is presented. In ActiWeb we also provide mobile code abstractions and scripting, but additionally a tight integration with the web and object-orientation is provided.

In [3] several means for building a web object model on the available web means is discussed. These include XML, RDF, DOM, embedded scripts, and simple RPC messaging techniques. There are several approaches for interaction using RPC based calls on the web, like WIDL [19], XML-RPC [20], or WebBroker [6]. Those send their data using an XML encoding. However, these approach misses several important parts of ActiWeb, like the integration with the scripting language for rapid customization and component integration, the persistence service, and code mobility. Moreover, integration of the provided services is rather low-level. In contrast, ActiWeb provides one language model, paradigm integration, and services as dynamically loadable components.

ZOPE [21] is an object-oriented development environment for web pages that is based on the object-oriented scripting language Python. It also contains an object-oriented database and a web server. Web documents are treated as objects that can have active parts through the document template markup language (DTML). Object calls are also mapped to URLs. The approach of ZOPE is quite similar to ActiWeb for web site development, has currently a better development environment with rich integration facilities predefined, but it lacks important services, like registry, mobile code, metadata representation, etc.

There are several other application servers and document management systems integrated with scripting (and some of the other presented services), including AOL Server, Web Shell, Vignette V/5, or [5]. These system offer some service not available in ActiWeb, as for instance document management functionalities. However, many important base services of ActiWeb, like code mobility or integration with object-orientation, are missing.

5 Conclusion

As described in the previous section, there are several systems which implement partial aspects of our ActiWeb system, but lack other parts completely. In contrast to many current commercial middleware systems, like distributed object systems or application servers, ActiWeb provides extensibility, is based on web standards, enables several web representations, is integrated with an object-oriented scripting language, is extensible with C/C++ components, and solves customizability and performance problems by providing code mobility. Moreover, ActiWeb has integrated some of the most prevalently needed service for distributed web applications: Integration with HTTP communication, object persistence, flexible data representation through XML, metadata through RDF, and registry services.

The major benefit of using an integrated system with these feature is that is provides a single framework with a single language model for integrated development of distributed systems. Through the extensive use of web standards it enables principle interoperation with the most programming languages and platforms. The component model of the XOTcl language enables legacy integration and integration with applications written in other languages, such as C/C++. I.e., XOTcl handles the paradigm and language integration issues, so that the web developer has only to deal with the object-oriented paradigm and with one language model. Application parts, written in other languages, are integrated as dynamically loadable components. The component-based approach lets web applications be dynamically extensible with new services or service versions. The scripting language allows for flexible component glueing and eases construction/manipulation of string-based web content. Thus we gain high customizability of web applications.

The integration with an active web object system lets us provide several representations for one business logic. Since the integration is hidden in the place's web server, the user on the web browser does not see any difference to usual web pages.

As we have discussed in the related work section, there are still several services e.g. known from commercial products, missing in ActiWeb, like transactions for persistent objects, scalability issues, integration with several distributed object technologies and DBMSs, etc. Moreover, there are several open issues in the implemented components, as for instance some open security issues in the mobile code component.

XOTcl and the ActiWeb components are freely available from `http://www.xotcl.org/`.

References

[1] P. Ciancarini, R. Tolksdorf, F. Vitali, D. Rossi, and A. Knoche. Coordination multiagent applications on the WWW: A reference architecture. *IEEE Transactions on Software Engineering*, 24(5), 1998.

[2] J. Bosak. XML, Java, and the future of the web. http://sunsite.unc.edu/pub/sun-info/standards/xml/why/xmlapps.htm, 1997.

[3] F. Manola. Technologies for a web object model. *IEEE Internet Computing*, 3(1):38–47, January/February 1999.

[4] J. K. Ousterhout. Scripting: Higher level programming for the 21st century. *IEEE Computer*, 31(3):23–30, March 1998.

[5] A. Shah and T. Darugar. Creating high performance web applications using Tcl, display templates, XML, and database content. In *Proceedings of the 6th Annual Tcl/Tk Conference*, San Diego, California, September 1998.

[6] J. Tigue and J. Lavinder. Webbroker: Distributed object communication on the web. http://www.w3.org/TR/1998/NOTE-webbroker, 1998.

[7] G. Neumann and U. Zdun. XOTcl, an object-oriented scripting language. In *Proceedings of Tcl2k: The 7th USENIX Tcl/Tk Conference*, Austin, Texas, February 2000.

[8] W3C. Document object model (DOM) level 1 specification. http://www.w3.org/TR/REC-DOM-Level-1, 1998.

[9] G. Neumann and U. Zdun. Enhancing object-based system composition through per-object mixins. In *Proceedings of Asia-Pacific Software Engineering Conference (APSEC)*, Takamatsu, Japan, December 1999.

[10] G. Neumann and U. Zdun. High-level design and architecture of an HTTP-based infrastructure for web applications. *World Wide Web Journal*, 3(1), 2000.

[11] G. Neumann and U. Zdun. Pattern-based design and implementation of an XML and RDF parser/interpreter. Submitted for publication, 2000.

[12] O. Lassila and R. R. Swick. Resource description framework (rdf): Model and syntax. http://www.w3.org/TR/WD-rdf-syntax/, 1998.

[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[14] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Patterns for Concurrent and Distributed Objects*. Pattern-Oriented Software Architecture. J. Wiley and Sons Ltd., 2000.

[15] S. Vinoski. Corba: Integrating diverse applications within distributed heterogeneos environments. *IEEE Communications Magazine*, 14(2), 1997.

[16] J. White. Mobile agents white paper. http://www.genmagic.com/technology/techwhitepaper.html, General Magic, Inc., 1995.

[17] R. S. Gray, D. Kotz, G. Cybenko, and D. Rus. D'Agent: Security in a multiple-language, mobile-agent system. In G. Vigna, editor, *Mobile Agents and Security*, volume 1419 of *LNCS*. Springer-Verlag, 1998.

[18] D. Kotz and R. S. Gray. Mobile agents and the future of the internet. *ACM Operating Systems Review*, 33(3), August 1999.

[19] M. G. Wales. Widl: Interface definition for the web. *IEEE Internet Computing*, 3(1):55–59, January/February 1999.

[20] XML-RPC home page. http://www.xml-rpc.com/.

[21] A. Latteier. The insider's guide to Zope: An open source, object-based web application platform. *Web Review*, March 1999.