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 realized with a diverse set of technologies, which are orthogonal to web technology and based on overlapping concepts, abstractions, and paradigms. In this paper we present* ActiWeb *as a single framework which 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 glueing. Moreover, some basic services, such as 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, 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.

*Published in: Proceedings of the 2nd International Conference on Internet Computing (IC'2001), Las Vegas, Nevada, USA, June 25–28 2001

But this simple architecture has several drawbacks for distributed web application development: It lacks support for interactive or collaborative multi-user applications [2] 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 [1]. 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 [6], 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, say, with the CGI interface, is often complex and misses the flexibility and customizability required by many web applications (see [11, 12]). 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 [13].

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 of active objects is not new, it has been used in the context of other domain. However, our component framework is implemented with today's web technology without suffering from the complexity of e.g. middleware approaches. It also avoids the limitations of CGI-like architectures, such as problems with client-to-client interaction, customizability, performance, etc.

2 A Component Framework for Active Web Objects

This section presents the basic architecture of our ActiWeb component framework. In this section, we give a very brief overview of the object-oriented scripting language XOTcl in which ActiWeb is implemented. XOTcl provides a set of uncommon functionalities that ease the implementation of our component framework. Then we discuss the conceptual hierarchy of the basic web object types 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) [9] (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, 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. Besides a highly flexible object system, XOTcl 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 "activeness" for web artifacts. A web object is used as a unified entity for application development and information exchange on the web. Web artifacts are represented by an object that contains methods defining the web artifact's behavior. In this context, the traditional web with its interconnected web pages can be interpreted as a drastically simplified system of web objects. Web pages can be seen as objects unambiguously identified by URLs. Links can be seen as methods invoked via a web server, which functions as dispatcher.

The notion of web objects also manifests in variants of dynamic HTML and the Document Object Model (DOM) [16] which treat HTML documents as programmable objects. In such approaches, the programmer can interact with document elements without parsing the text again. However, DOM based approaches only offer limited suitability for general application development, since they are conceptually centered around classical web documents. Thus resources for expressing the application semantics of distributed systems are not provided.

In our concept, 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, such as 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.

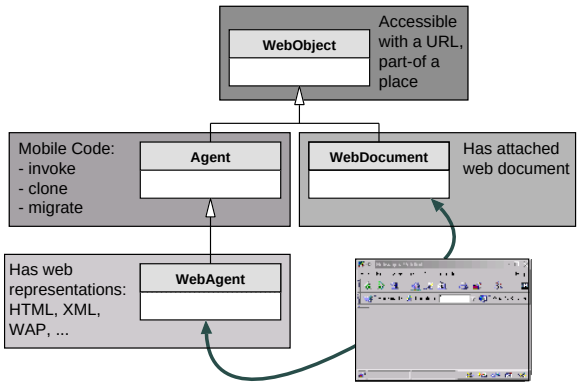

Figure 1: Conceptual Class Structure of Web Objects

An important property of a web object is its location. An active web object "lives" in the runtime environment of a place. The place 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.

An object that can be accessed through an ActiWeb place is called *web object*. Each *place* is uniquely identified by host name and port number of the HTTP server. Additionally, each web object has a string-based object name as object ID. 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 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 dynamically and transparently attach a class to an arbitrary object as an implementational role using per-object mixins [7]. Therefore in ActiWeb an agent object can behave exactly like a web document and vice versa.

ActiWeb integrates mobile code technology and object-oriented programming with the current web infrastructure. The *place* abstraction of the mobile code paradigm is a good access point for the active web object system as well. It also serves for all centralized issues, such as integration of service components, the mapping of URLs to methods of web objects, security and access control issues, etc. 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 2) is implemented on top of XOTcl. Each service is a dynamically loadable component. Generally all components can be substituted by compatible ones. xoComm [8] provides an object-oriented implementation of an HTTP server and HTTP access. Places, the basic execution environments of ActiWeb, contain exactly one HTTP server identified by host and port. Web objects can access other (remote) web objects via HTTP.

The data representation services provide an object-oriented implementation of an XML- (called xoXML) and an RDF-parser/-interpreter (called xoRDF). xoStore is a general persistence service for XOTcl objects. A registry service xoReg enables registration of ActiWeb objects, say, to find an object through the specification of certain properties. xoMOS implements an mobile object system. 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.

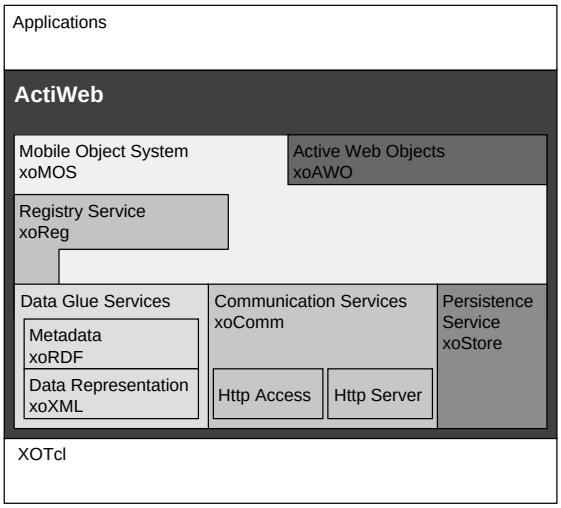

Figure 2: ActiWeb: Basic Architecture

3.1 Flexible Data Representation and Metadata Services

For flexible data representation an XML and RDF parser/interpreter framework is integrated in ActiWeb [10]. XML is primarily used as a flexible data glue and a platform independent data representation. The Resource Description Framework [4] is a formally specified model for describing 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 in 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 also special metadata properties of agents, such as migration properties. 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. When agents migrate, they generate such RDF metadata.

3.2 HTTP-Based Communication Service

xoComm [8] 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-side implementation for providing the RPC communication resources for their agents.

In ActiWeb each place aggregates a web server object. Per default all place communication is handled by the place's web server. Each ActiWeb agent is able to invoke, clone, and migrate itself via HTTP. Agents generally exploit asynchronous communication. Agents must register themselves with a place. 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

Object persistence is required for most distributed web applications. A persistence service enables recovery of web objects and agents from non-volatile storage, say, for cases when a place has a fault or is turned down temporarily. If an object persistence service is missing, it usually has to be programmed by hand, say, on top of a relational database.

The xoStore persistence service allows the developer to make any XOTcl object persistent with a single line of code. It realizes different storage Strategies which are mixed-in as per-object mixins [7]. Different backends, like various persistent stores, can transparently be used with a unique interface. Objects that need persistence may add persistence transparently and dynamically through Per-Object Strategies. Currently clients can choose between an eager strategy and a lazy strategy. The eager strategy writes changes in the object's data to the storage as they occur. The lazy strategy writes the object's data when the process terminates.

3.4 Mobile Web Object System

The components of the mobile object system implement a remote programming (RP) and remote procedure call (RPC) environment for mobile agents. To let a web object 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 unambiguously identified by host name and port. All web objects are, therefore, also unambiguously identified 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 an 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 themselves with the agent manager. It (lazily) creates RDF metadata on the agent's 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 with the ability to `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, encodes 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 start command specifies 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 RDF metadata, like type, name, attributes, etc.

Other agents can send a request for properties to the registry. 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. Several registry agents in different places may be connected and can forward queries to other registries, say, in a hierarchical fashion (similar to the domain name service (DNS)).

3.6 Web Representations for Active Web Objects

In many distributed applications there are diverse requirements for communication and representation. Often it is hard to predict all required representation and channel types for interaction. Active web objects in the ActiWeb system are an extensible form of representations via different channels, such as RP, RPC, SOAP, CORBA, HTTP, WAP, etc.

An active web object can provide web representations for 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 web agents that contain a web document. That way a document can 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 hide the representation. Thus, a Service Abstraction Layer [15] providing views for different channels is used. The place acts as a Proxy and forwards calls for a special representation to the proper Facade.

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. Figure 3 presents the provided basic communication resources and channels of the ActiWeb system, yet they can be extended with other channels and representations, such as SOAP, CORBA, or WAP.

4 Related Work

The commonly used CGI interface has several drawbacks in comparison with the presented architecture. CGI applications are often a loose coupling of scripts without a component concept. Thus complex applications may be hard to maintain. Most tasks have to be programmed by hand instead of using a simple 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 ActiWeb to a distributed object system, like CORBA [14] or DCOM, we can assess that both solutions are able to hide the networking details. Some of these approaches, like CORBA 3.0 or Java RMI/EJB, also offer a component model integrated with distributed objects. However, middleware solutions do not integrate well with scripting and the web. Thus solutions may be more complex. The distributed object system usually does not provide an integrated solution for static and active web rep-

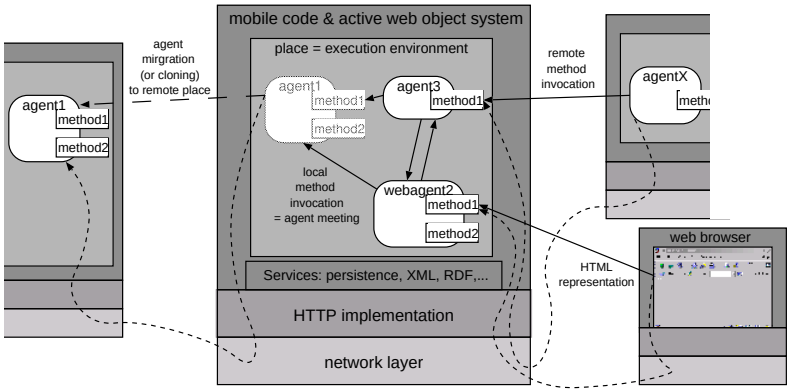

Figure 3: Mobile Code and Active Web Object System – Communication Resources and Channels

resentations. The usage of a conventional application server, such as WebLogic, WebSphere, etc., offers a concept for adding an additional web representation to a business logic. But the concept does not ease the use of other representations. As a pure server-side approach it may produce a significant network load for client-to-client interaction.

Nevertheless, commercial products, like distributed object systems 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 be integrated. Furthermore, for interoperation 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 [18] 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 as well. Telescript does not support web representations and is not integrated with web standards. In [3] 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 [6] several resources for building a web object model on top of available web resources 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 [17], XML-RPC [19], or WebBroker [13]. Those send their data using XML encoding. However, these approaches miss 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 [5] 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 services presented), including AOL Server, Web Shell, Vignette V/5, or [12]. These systems offer some service not available in ActiWeb, as for instance document management functionalities.

However, many important basic 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. 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.

Despite the rich service environment, only a very limited number of components with small interface has to used by application developers. For instance, all XML/RDF encoding/decoding is handled automatically. Thus the direct use of XML/RDF is optional. In other words, the basic requirements to start development in ActiWeb can be learned quite quickly and applications are usually less complex than applications written entirely in system languages.

For many typical web applications, ActiWeb can be used as a single framework for integrated development of distributed systems. The component model of the XOTcl language enables legacy integration and integration with applications written in other languages, such as C/C++. Application parts, written in other languages, are integrated as dynamically loadable components. 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 client does not see any difference to usual web pages.

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

References

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

[2] 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.

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

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

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

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

[7] 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.

[8] 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.

[9] 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.

[10] G. Neumann and U. Zdun. Pattern-based design and implementation of an XML and RDF parser/interpreter. Submitted for publication, 2001.

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

[12] 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.

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

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

[15] O. Vogel. Service abstraction layer. submitted to EuroPLoP 2001, see also http://www.ovogel.de/SAL.htm, 2001.

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

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

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

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