Active Hypertext for Distributed Web Applications

Eckhart Köppen, Gustaf Neumann
{Eckhart.Koeppen,Gustaf.Neumann}@uni-essen.de
University of Essen, Information Systems and Software Techniques
Universitätsstraße 9, 45141 Essen, Germany

Published in: Proceedings of The Eighth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET-ICE'99),
June 16-18, 1999, Stanford.

Abstract

The prevailing architecture for web-based applications relies on HTML, HTTP and loosely integrated functional elements, propagating a strong distinction between client and server. The proposed approach for active hypertext documents however realizes applications through hypertext documents with embedded application logic. Furthermore, the need for separating the approach into a model, a development method and supporting tools is identified. The core model is based on open standards such as XML and CSS. It is extended by idioms which describe important concepts and reusable constructs. Examples for the application of the model are given.

1 Introduction

Undoubtedly, the World Wide Web is the most dominant of the Internet techniques to date. Using two simple mechanisms (HTML and HTTP, [22] and [9]) a very wide range of applications has been implemented. Among those are large-scale distribution and retrieval of information or electronic commerce. The process of implementing the web-based information systems behind these applications has however shown the deficiencies of the existing mechanisms and the need for the specification and implementation of new mechanisms. One of the new mechanisms which was introduced are *active hyperlinked documents* [13], using XML [6] and other Web and Internet standards to combine data, behavior, state and presentation. This paper wants to further elaborate and discuss the proposed model and show how active documents can be used to construct web-based applications.

2 Why Active Documents?

The application domain of web techniques (mainly HTML and HTTP) has changed over time from information distribution and authoring to the implementation of *web-based information systems*, i.e. systems that provide functionality similar to conventional information systems. They typically follow a very simple client-server model: A web server distributes information (HTML documents) and allows its modification through predefined actions (CGI [8] scripts), while the web clients are only used to display the distributed information and invoke the available actions.

Looking at these systems, two aspects become obvious: First, there is an imbalance between the tasks web clients on the one hand and web servers on the other hand have to fulfill. Second, the underlying processing and data model is very simple, based on information structuring using HTML and information processing by CGI scripts.

While the first aspect has been dealt with through extensions on the client side (mostly Java [11] and JavaScript [20]), appropriate and applicable solutions which cover the second aspect are still not available. Though it is possible to realize a sophisticated processing model using Java, approaches to provide an equally sophisticated information model are missing. Besides, the advantages of hypertext systems like the WWW such as easy information distribution and authoring are lost the more the implemented systems come close to distributed systems based on CORBA [21], RPC [25] or Java RMI [26].

This leads to the idea of *active documents*. An active document combines structured and hyperlinked information like any other hypertext document, but it also contains associated procedural information in the form of scripts, program fragments or references to processing instances accompanied by state information. The applications which can be realized using active documents cover a very wide range: simple standalone applications like smart bookmarks, cooperative applications using hyperlinked documents like intelligent catalogs or applications involving mobile active documents to implement an agent system. In this paper, a calendar for a workgroup will be used as an example. Before, a more precise definition of the underlying model is given.

3 The Active Hypertext Document Model

The basic idea of combining code and data will be refined through the definition of the *Active Hypertext Document Model* (AHDM). It builds upon the previously referenced work in [13].

3.1 Information Model

The information carried by an active document is subdivided into the following classes:

Behavior: Information which is used to describe behavior is mostly made up of suitably sized units like subroutines written in scripting languages. The scripts are closely tied to the AHDM, i.e. in that context, they are an essential part of the document. Outside the context of the AHDM, they are not considered to contribute to the information carried by the enclosing document.

State: Like the information used to describe behavior, the state of an active document contributes to a distinct class of information. It contains the information which only serves the purpose to control the behavior of an active document. This is in contrast to the general data contained in a hypertext document (see below). The state information is only used in the AHDM and can be removed from the document in other contexts.

Presentation: The presentation of a document is controlled by dedicated presentation information. Similar to the description of state and behavior, the presentation information is only considered an essential part of a document if it is used in an appropriate context. Here, this context is the consumption of a document by end users.

Linking: Assuming uni-directional linking, the information used to link two documents together consists of three parts: The starting point or anchor within the source document, the target of the link (either a whole document or parts of it) and additional link information like invocation semantics. The linking information is part of the source document. Because of the importance of hyperlinking not only for the AHDM, it is an essential part of the document in any context.

Data: Any information not used to describe behavior, state or presentation contributes to the class of general data. The linking information plays a special role here, because it may overlap with the general data. The general data within a document is the information which is not explicitly tied to the AHDM, though the behavior of an active document is not only dependent on the state information but can also refer to and modify general data.

3.2 Elements of the AHDM

The basic elements are those parts of the AHDM which can be generally used in conjunction with hypertext documents:

- *XML* and *XML Namespaces* [7]: Information structuring and combination
- *XLink* [16] and *XPointer* [17]: Information linking
- *Cascading Style Sheets* [15]: Information presentation
- *Document Object Model* [29]: Programming Interface for document manipulation
- *Intrinsic Event Model* [22]: Connection of external events to behavior descriptions
- *HTTP*: Information distribution and remote method invocation
- *Scripting Languages*: Description of behavior

3.3 AHDM Schema

The most important element of the AHDM is the AHDM schema or namespace. It contains the declaration of XML elements needed to construct an active hypertext document. Following the requirements, two elements are defined in the namespace: *Behavior* is described using the *func* element and *State* is held in *var* elements. To provide greater *flexibility and scalability*, a third element is introduced: The *delegate* element denotes a link to a remote document which can be used to delegate behavior. This will be clarified in the section on the runtime environment. The resulting DTD consists of the following declarations:

```
<!ELEMENT var (#PCDATA)>
<!ATTLIST var
            NAME CDATA #REQUIRED>

<!ELEMENT func (#PCDATA)>
<!ATTLIST func
            name CDATA #REQUIRED
            name CDATA #REQUIRED>

<!ELEMENT delegate EMPTY>
<!ATTLIST delegate
            xml:link CDATA "SIMPLE" #FIXED
            actuate CDATA "auto" #FIXED
            show CDATA "new" #FIXED>
```

Figure 1: The AHDM DTD

3.4 Runtime Environment

The runtime environment exports the main AHD support functions which can be used from within the program fragments in an AHD. In short, these functions cover the following aspects: Invocation of *func* elements (call), query and modification of *var* and other elements (get, put), loading remote documents (load) and obtaining a persistent version of the current active document (toText).

All AHD elements are referenced through their name attribute. Furthermore, the type attribute of the *func* element denotes the MIME type [5] of the embedded script.

The runtime functions enforce a certain document architecture through *parent delegation*. In this architecture, elements with sub-elements from the AHDM namespace are similar to objects in the respect that all sub-elements are considered to be in the same scope. If for example an element with the general ID article has the sub-elements with tag names *func* or *var*, these sub-elements are associated functions and variables for the article element. If another function element is to be invoked from one of these sub-elements, the lookup for the called element starts at the direct parent, considering all direct children of this element. If the called element is not found, the lookup continues at the grandparent of the calling function element until the root element is reached. In the example shown below, the function update_total is first searched within the scope of the article element and then in the scope of the invoice element.

```
<invoice> ⇐ 2nd scope
   <func name="update_total"> ... </func>
   <article> ⇐ 1st scope
      <func name="print"> ... </func>
      <func name="add_item">

         ... this function invokes update_total

      </func>
      <var name="status"> ... </var>
      ...
```

Figure 2: Parent Lookup

In addition to parent delegation, *remote delegation* is possible through the *delegate* element. This is a simple XLink, where the HREF attribute references a remote active document. The runtime environment follows this link if an element cannot be found in the parent chain of a referring element. In the case of function invocations, the referenced function will be invoked in the remote document if it cannot be found in the local document. This works likewise for variable access.

func elements can be invoked either locally (following parent delegation) or remote. The remote invocation is realized through a HTTP POST request. Parameters are passed in the body of the POST request and will be decoded into named parameters on the receiver side. Variables and other resources like remote XML entities and documents are accessed through the HTTP GET and PUT elements.

The interface to the runtime environment is defined by a IDL interface (using the DOM IDL interfaces):

```
interface AHDRuntime {
   void put (in Element current,
                in DOMString varName,
                in DOMString varValue);

   DOMString get (in Element current,
                in DOMString varName);

   DOMString call (in Element current,
                in DOMString funcName,
                in DOMString paramString);

   Document load (in DOMString urn);

   DOMString toText (in Document doc);
};
```

Figure 3: Runtime Interface

The AHDRuntime interface exports methods for variable access (*get* and *put*) as well as the *call* function for method invocation. A remote document is transferred to the local runtime via the *load* function which fetches and activates the document. The *toText* function can be used to store or transmit a textual representation of an AHD.

A sample runtime environment is currently implemented as part of the *Kino* XML/CSS processor [14].

4 A Sample Application

To show the potential of active documents and to point out some differences to traditional approaches, a sample application involving a distributed calendar shall be presented here. The scenario is characterized as follows:

- A group of people coordinates appointments via a shared calendar. An appointment is defined through start and end times, title, location and participants.
- The shared data is held at multiple locations (i.e. every user has a local copy of the calendar data).
- Appointments are made and modified by sending the appointment data from the person who wishes to arrange or change an appointment to any of the other group members.

An application built using active documents could have the following architecture:

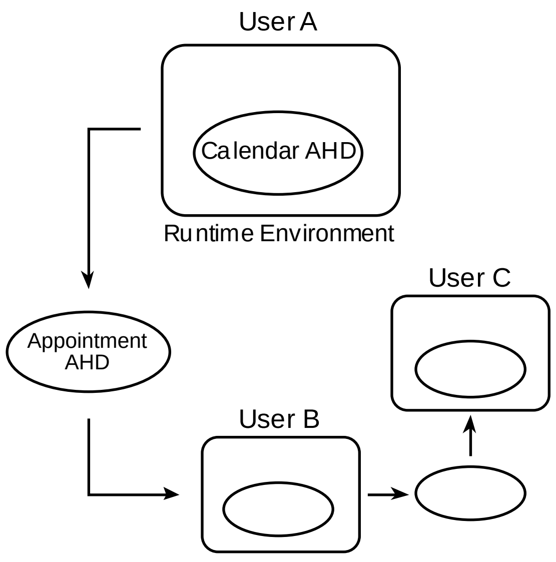


Figure 4: Application based on Active Documents

The application is solely based on independent documents which communicate with each other: the calendar data is stored in an active document within the runtime environments and new or changed appointments are transmitted as active documents between the runtime environments of the group members.

The most important parts of the appointment and calendar document are shown in Figure 5 and Figure 6 (note that most elements have default attributes assigned in DTDs not included here). Figure 5 shows a sample code fragment for a single appointment, containing start and end times and a participant list.

```
<appointment>
   <id>0</id>

   <field>
      <label>Start: </label>
      <starttime>05/11/99 12:30</starttime>
   </field>

   <field>
      <label>End: </label>
      <endtime>05/11/99 14:00</endtime>
   </field>

   <particiantlist>
      <label>Participants: </label>
      <participant calendar="urn:Dridi:calendar">
         Dridi
      </participant>
      …

   </participantlist>
   …

</appointment>
```

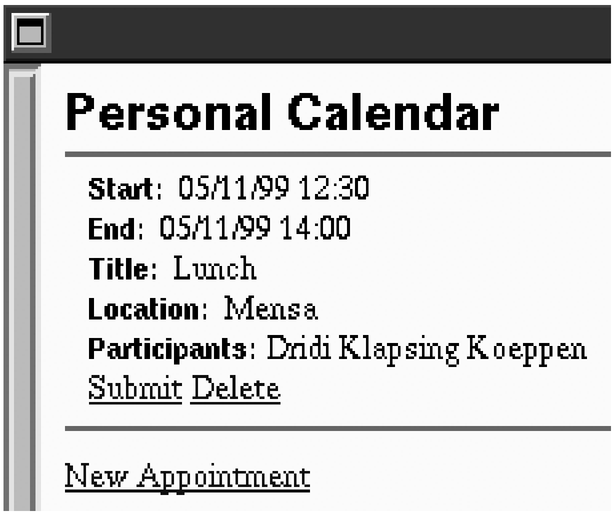
Figure 5: Appointment Document

In Figure 6, the container document for the appointments is shown. The appointments are embedded in an appointment list, furthermore, *func* elements are used to implement the calendar functionality.

```
<calendar>
   <func name="new_appointment">…</func>

   <func name="submit_appointment">…</func>

   <func name="delete_appointment">…</func>

   <heading>Personal Calendar</heading>

   <appointmentlist>
      <appointment>
         …
      </appointment>

      …
   </appointmentlist>
   …
</calendar>
```

Figure 6: Personal Calendar Document

Figure 7 shows a screenshot of a personal calendar. The document provides four functionalities: the ability to change appointment data like the start time or the title, submission of this changed data to the other group members, deletion of an appointment and creation of a new appointment. The functions are invoked through IEM events, in this case the onclick event which is handled by the HTML link-styled words or by the labels for the appointment data.



Personal Calendar

Start: 05/11/99 12:30
End: 05/11/99 14:00
Title: Lunch
Location: Mensa
Participants: Dridi Klapsing Koeppen
Submit Delete

New Appointment

Figure 7: Personal Calendar Screenshot

It is important to note that the distinction between client and server is practically removed because the original functionality of the server (e.g. providing an CGI interface to a database, executing CGI scripts) is now implemented in the active documents and can also be utilized in the clients. Additionally, the documents may be used in place of databases, a field where XML shows promising features. Though the above example utilizes a decentralized approach, it would also be possible to use a central monitor document which coordinates the application Figure 8 shows a fragment of the personal calendar where a DELEGATE element at the root of the calendar document is used to delegate behavior (i.e. function calls) to a central document, and the method call in the submit_appointment function, where the appointment data is passed as an XML string to the change_appointment function. The programming language used here is Tcl, but other languages with an extensible DOM binding could be used as well.

```
<calendar>
   <delegate href="urn:SWT:monitor">
   …
   <func name="submit_appointment" type="text/tcl">
      …
      ahd::call change_appointment \
         appointment $text
   </func>
   …
</calendar>
```

Figure 8: Personal Calendar Document.

The behavior of this document can now easily be changed, for instance if the delegate document is not reachable, by simply modifying the reference to the delegate document. This is also described in the next section on idioms and patterns.

5 Idioms

To enhance the usability of a model like the AHDM, Idioms can be used. Idioms describe recurring themes and name them with a simple term, patterns offer predefined, customizable solutions to recurring problems, differing only in the scale of their applicability. In the above example, a number of idioms can be identified which may also be found used in other applications. Idioms can be encapsulated and formalized using patterns.

Activity: The current architecture of web-based information systems implies that active hypertext documents generally have two states: They are either filed in a database or file system, thus *inactive,* or they are loaded into some kind of web or application server, where they are *active*. The transition between the two stages is only triggered by external events: Requests for an AHD or parts thereof, invocation of a *func* element in an AHD or explicit saving of an AHD to the file system upon the termination of the runtime environment. An AHD is notified about the state changes through the IEM events *onload* and *onunload*.

Persistence: Very closely tied to the notion of activity is the concept of *persistence*. An active hypertext document is made persistent by making it inactive and saving it to some kind of persistent storage. It is very important to note that only the parts of an AHD are considered to be persistent that are part of the document's structure. This includes all XML elements, their attributes and their contents, but it excludes variables of the program fragments in the *func* elements.

Reference: The reference to an AHD is made through *Uniform Resource Identifiers* which can have a wide range of semantics. The most prominent URI is the URL which describes a physical location for a document. The idiom of a *change of reference* denotes in most cases a change of the physical location of an AHD. The AHDM does not propose any explicit mechanisms to deal with those location changes, but it is clear that they can result in problems like link consistency. Generally, no assumptions about the meaning reference to an active document shall be made. This might change with a centralized name service that can map meaningful URIs or URNs to physical location identifiers.

Modification of Behavior: An interesting aspect which results from the concept of *parent delegation* is the ability to change the behavior of an AHD. The rules for the element lookup make it possible to implicitly refer to AHD elements in the parent chain of an element. An element can therefore be moved into a different context without changing the associated behavior descriptions. Equally, the context of an element can be changed. With this approach, it is possible to build libraries of independent elements which can be reused in other documents. The concept of remote delegation even allows for flexible reconfiguration of behavior by changing the delegate documents.

It is interesting to note that the above idioms result mostly from the change of the properties or state of an active document, e.g. activation or deactivation or reference changes.

6 Related Work

Work related to active documents can be found in various areas. Besides techniques to be used in web-based systems (see for example the approaches concentrating on the client side like [1], [12], [18], [20] and [24], or on the server side like [27] or [28]), other approaches exist in electronic publishing [2], workflow systems [3] or mail systems ([4], [10]). In this paper however we concentrate on work that can complement the AHDM directly or indirectly to make clear that the AHDM is intended to integrate other models and techniques and not to compete with them.

While many of the above approaches consider only single aspects of the implementation of web-based information systems, the AHDM covers multiple aspects by removing the distinction between client and server and by providing an information model. However, many of the alternative approaches can be integrated into or used in conjunction with the AHDM. The AHDM can provide a uniform, document-centric interface for other information and active resources on the web. It can serve the role of an integrating middleware, making technologies that can complement the AHDM in this respect interesting. Among those technologies is for example CORBA: An active document could be exported by the runtime environment through a CORBA object request broker. This is facilitated by the document structure where any element with sub-elements from the AHDM namespace can be viewed as objects with methods and state variables. Likewise, it would be possible to implement CORBA services over HTTP to reference CORBA objects transparently using the AHDM runtime from within an AHD, comparable to the approach in [19].

7 Future Directions and Conclusion

As a result of separating the different parts of an active document, a set of *reusable components* (mostly style sheets and behavior descriptions) can evolve. They can be gained by factoring out generally usable parts of a document. In a context where the structure of the documents play an important role, the reusable components include schema definitions (e.g. DTDs) as well. The components can be reused by simply linking them to the base documents, supporting truly distributed authoring.

The architectural approach which was shown above can be used to implement computer supported workflow systems on a larger scale. It is based on the notion of *personal repositories*, where active documents are stored. They are accessed using personal web servers and are associated to one or more persons. This architecture relies on the migration of documents between repositories and the implementation of a workflow logic inside the documents.

The AHDM provides a stable core for implementing autonomous, active documents which can be linked together to form web-based applications. However, a development method still needs to be defined to enforce certain software quality characteristics such as flexibility or reusability. Other issues not dealt with in this paper include security aspects. They can be addressed on multiple levels: Implementing security mechanisms at the application level, in the runtime environment or at the document level.

References

[1] S. Ball: *Embeddable Components For Stand-Alone Web Applications*, Proc. of AusWeb97, Lismore 1997.

[2] E. A. Bier: *Documents as User Interfaces*, Proc. of the Intl. Conference on Electronic Publishing, Document Manipulation and Typography, 1990.

[3] R. Bentley, T. Horstmann, K. Sikkel, J. Trevor: *Supporting Collaborative Information Sharing with the World Wide Web: The BSCW Shared Workspace System*, Proc. of the 4th World Wide Web Conference, Boston 1995.

[4] N. Borenstein: *Computational Mail as Network Infrastructure of Compter-Supported Cooperative Work*, Proc. of the Conference on CSCW, Toronto 1992.

[5] N. Borenstein and N. Freed: *Multipurpose Internet Mail Extensions*, Standards Track Protocol, RFC 1521, September 1993.

[6] T. Bray, J. Paoli, C.M. Sperberg-Queen: *Extensible Markup Language (XML) 1.0*, W3C Recommendation, , February 1998.

[7] T. Bray, D. Hollander, A. Layman: *Namespaces in XML*, W3C Recommendation, January 1999.

[8] K. A. L. Coar, D. R. T. Robinson: *The WWW Common Gateway Interface Version 1.1*, Internet Draft, May 1998.

[9] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee: *Hypertext Transfer Protocol - HTTP/1.1*, Standards Track Protocol, RFC 2068, January 1997.

[10] Y. Goldberg, M. Safran, E. Shapiro: *Active Mail - A Framework for Implementing Groupware*, Proc. of the Conference on CSCW, Toronto 1992.

[11] J. Gosling, B. Joy, G. Steele: *The Java Language Specification*, Addison Wesley Longman, Reading 1996.

[12] M. F. Kaashoek, T. Pinckney and J.A. Tauber: *Dynamic Documents: Extensibility and Adaptability in the WWW*, Proc. of the 2nd World Wide Web Conference, Chicago, 1994.

[13] E. Köppen, G. Neumann: *A Practical Approach towards Active Hyperlinked Documents*, Proc. of the 7th World Wide Web Conference, Brisbane 1998.

[14] E. Köppen, G. Neumann: *Implement, Gustaf! – The Kino XML/CSS Processor*, Poster Proc. of the 8th World Wide Web Conference, Torono 1999.

[15] H. W. Lie and B. Bos: *Cascading Style Sheets, level 1*, W3C Recommendation, December 1996.

[16] E. Maler, S. DeRose: *XML Linking Language (XLink)*, W3C Working Draft, March 1998.

[17] E. Maler, S. DeRose: *XML Pointer Language (XPointer)*, W3C Working Draft, March 1998.

[18] D. Massy, S. Williams, R. Norlander, L. Kurata, T. Reardon: *HTML Components*, W3C Note, October 1998.

[19] P. Merle, C. Gransart, J.-M. Geib: *CorbaWeb: A Generic Object Navigator*, Proc. of the 5th World Wide Web Conference, Paris 1996.

[20] Netscape Communications Corp.: *JavaScript Reference*, December 1997.

[21] Object Management Group: *The Common Object Request Broker: Architecture and Specification*, August 1997.

[22] J. K. Ousterhout: *Tcl: An embeddable Command Language*, Proc. of the USENIX Winter Conference, January 1990.

[23] D. Raggett, A. Le Hors, I. Jacobs: *HTML 4.0 Specification*, W3C Recommendation, April 1998.

[24] R. Stevahn: *Adding Style and Behavior to XML with a dash of Spice*, W3C Note, February 1998.

[25] Sun Microsystems: *RPC: Remote Procedure Call Protocol Specification Version 2*, RFC 1057, June 1988.

[26] Sun Microsystems: *Java Remote Method Invocation - Distributed Computing for Java*, May 1998.

[27] Sun Microsystems: *Java Servlet API Specification*, November 1998.

[28] R. Wodaski: *ASP Technology Feature Overview*, August 1998.

[29] L. Wood et al.: *Document Object Model (DOM) Level 1 Specification*, W3C Recommendation, October 1998.