ARCHITECTURE TO CONNECT TOOL-BASED WEB INTERFACES TO SERVICE-ORIENTED ARCHITECTURES

Claus-Peter Klas, Martin Mois and Matthias L. Hemmje
FernUniversität in Hagen, Germany

Keywords: Digital library, Web 2.0, Information retrieval, User interface, Ajax.

Abstract: DAFFODIL is a user-oriented digital library system for accessing distributed libraries through a unique user interface. Beyond that, DAFFODIL also provides high-level functions to support proved search strategies which allow a system-wide search and navigation over the integrated digital libraries. The DAFFODIL system is based on distributed agents and comes with a Java Swing-Client which has to be installed on the user's system. This paper describes an architecture for combining the powerful services of DAFFODIL with a modern Ajax-enabled interface to lower the access boundaries for efficient and effective information search for all users.

1 INTRODUCTION

DAFFODIL[1] is a user-oriented digital library system for accessing distributed libraries (Schaefer et al., 2002). As the numerous digital libraries differ in functionality, search language and user interface, the DAFFODIL system provides unique access to these heterogeneous and federated sources. Beyond that, DAFFODIL also provides high-level functions to support proved search strategies which allow a system-wide search and navigation over the integrated digital libraries.

The service-oriented architecture (SOA) of the DAFFODIL system is based on distributed agents (Fuhr et al., 2000). The current graphical user interface is implemented as a Java Swing-Client which has to be installed via Java Webstart on the user's system as a standalone application. As many companies or universities restrict or even forbid the installation of software on a client machine, the pertained users are excluded from the usage of DAFFODIL.

Assuming that access to the internet using a modern web-browser is feasible on most client machines and therefore also on client machines which make an installation of DAFFODIL difficult, a web-based user interface would allow these users to utilize the functionality provided by the DAFFODIL system. To overcome the listed reasons, the development of an architecture for a web-based interface for DAFFODIL was considered. Using only standard JavaScript and the Ajax technology on the client-side is therefore an obvious solution, as support for this language comes as a built-in of nearly all modern web browsers.

As the DAFFOIL Java client is based on the WOB-model (Krause, 1997), a tool-metaphor based strictly object-oriented graphical-direct manipulative user interface, which has in many evaluations proven to raise efficiency and effectiveness of the users, the architecture for a web-based interface has also to support the tool concept allowing an easy addition or removal of functionality.

Thus, this paper presents an architecture aiming to fulfill the following requirements:

- Provide DAFFODIL's basic functionality over the web to lower the access boundaries for efficient and effective information search.
- Transfer the user interface concepts of the tool-based WOB model to the web and thereby solve the inherent problems of this approach.
- Allow an easy integration of further or new functionality and tools, which already exist in the DAFFODIL Java client.
- Connect a web application to DAFFODIL's SOA, thus combining message oriented communication with the web's client/server architecture.

The remainder of this paper is organized as follows. Section 2 introduces the DAFFODIL system and its architecture. In section 3 the develop-

[1]Distributed Agents for User-Friendly Access of Digital Libraries

Klas C., Mois M. and L. Hemmje M.
ARCHITECTURE TO CONNECT TOOL-BASED WEB INTERFACES TO SERVICE-ORIENTED ARCHITECTURES.
DOI: 10.5220/0001839103290334
In *Proceedings of the Fifth International Conference on Web Information Systems and Technologies (WEBIST 2009)*, page
ISBN: 978-989-8111-81-4

ment of an Ajax-based web application implementing DAFFODIL's WOB model is motivated in order to describe in section 4 our tool-based architecture. Section 5 comes up with some aspects of the implementation while section 6 discusses related work. Finally, section 7 gives a summary and lists some open issues.

2 DAFFODIL

Next to the seamless integration of different digital libraries and a system-wide search and navigation over them, DAFFODIL also aims to provide strategic support to the user, based on the concept of high-level search activities as described in (Schaefer et al., 2002). In order to implement these high-level functions an agent-based architecture was chosen. Each agent of this SOA implements a specific functionality like e.g. a wrapper for queries to a specific digital library or a thesaurus. Communication among the agents is based on a common XML format.

The current graphical Java frontend is implemented as an agent offering the existing functionality through a set of tools to the user. Each tool opens one or more views which are currently rendered as tabs on different window panes. An interaction with these views causes the GUI agent to ask the appropriate agents in the backend to execute the requested functionality. Thus, the combination of a SOA together with the user interface's tool concept provides an easy way of adding further functionality to the system.

3 MOTIVATION

The described architecture of the DAFFODIL system has proven to be extensible and scalable. But due to the wish to use DAFFODIL everywhere you are, despite issues like restricted user rights or a necessary installation process, a web-based user interface can extend the set of users and provide a basic set of DAFFODIL's functionality to the users on the internet. Emerging technologies like Ajax enable the implementation of DAFFODIL's user interface concepts as a web application.

3.1 Ajax

The abbreviation Ajax (Asynchronous JavaScript and XML) describes a web development technology which uses the script language JavaScript to load content asynchronously to the series of viewed web pages. Asynchronously downloaded content is then added to the currently displayed web page through an update of the page's DOM. Though the term Ajax was coined by an article written by James J. Garret (Garrett, 2005) in 2005, the underlying technologies JavaScript, Document Object Model (DOM) and XML have been used before to create more interactive web pages and to transfer proven user interface concepts from the desktop to the web.

3.2 The WOB Model

The WOB model for user interface design is based on the tool metaphor (Krause, 1997). It attempts to solve the inherent contradictions in the interface design process — like that between flexible dialog control and conversational prompting — using a set of co-ordinated ergonomic techniques.

The general software ergonomic principles of the *WOB* model are:

- *Strict Object Orientation and Interpretability of Tools*
  Strongly related functionality of the system is encapsulated in tools that are displayed as icons (not as menus). The tools open views, which are rendered as tabs on a window pane. Thus, the chain of views a user is working on can either be seen as a set of forms to be filled or as a set of tools that can be used to perform tasks more quickly.
  The user can manipulate objects on the surface in a direct manipulative manner. The model requires an object-on-object interaction style with a clear direction and semantic: To apply a function on an item, the latter has to be dragged to a tool.
- *Dynamic Adaptivity.*
  The interface adapts its layout and content always to the actual state and context. This is mostly used for a reduction of complexity in non-trivial domains.
- *Context Sensitive Permeability.*
  When known information is reusable in other contexts, it will automatically be reused.
- *Dialog Guideline.s*
  The views of the tools are functionally connected e.g. by means of action buttons, hypertext links or rules which are triggered by plan recognition. A tool can also open its view proactively if the user needs its function in a given situation.
- *Intelligent Components.*
  In order to decide if their function is valuable for the user, tools and controls in the interface have access to context and state. If applicable, they shall interact pro-actively with the user or the shared environment (the desktop), respectively.

Two principles of the model are information system-specific:

- *Status Display with Edit Mode.*
  The system shall always display a paraphrase of the current state for the user. It can be shown as a natural or formal language string or even by using some visual formalism (like a table).
- *Iterative Retrieval and Query Transformation.*
  Initial query formulations tend to be inadequate for the user's intentions, due to uncertainty or unconscious goals in the search process. Therefore applications shall simplify iterative query formulation for the user.

Technologies like Ajax and JavaScript make it possible to implement drag'n'drop mechanisms as well as dynamic adaptivity of the interface to the current state of each tool. In contrast to web applications that follow the REST style (Fielding and N., 2002) and that manage state on the server the principles "'Intelligent Components"' and "'Dialog Guidelines"' imply that a certain amount of state data is kept on the client in order to be able to react accordingly to user events and changes in context.

4 TOOL-BASED ARCHITECTURE

This section gives a detailed description of the developed tool-based architecture providing web-based access to DAFFODIL's SOA. First of all the vertical distribution of the components over client and server is discussed. In the following attention is turned towards the client/server communication and the detailed structure of client and server.

4.1 Vertical Distribution

The architecture to be developed has the aim to implement the WOB model. Beyond that it should also provide an easy to use extensibility mechanism for new functionality. We achieve these aims by using a XML-based communication together with an implementation of the publish/subscribe model. This also falls into place with the communication model used by DAFFODIL's SOA as therefore allows us to route messages received from the agent system directly to one of the tools.

Figure **??** shows four possible distributions of the components of the WOB model (view, tool, agent) to the two tiers web server and client. They are discussed in the following:

a) Splitted View. Possibility a) represents the classic web application model: the state of the application is held on the server side and in dependence of this state markup is created and sent to the client. While in this case the server has full control over the whole application, the client cannot make decisions on its own in order to react immediately to user interactions.

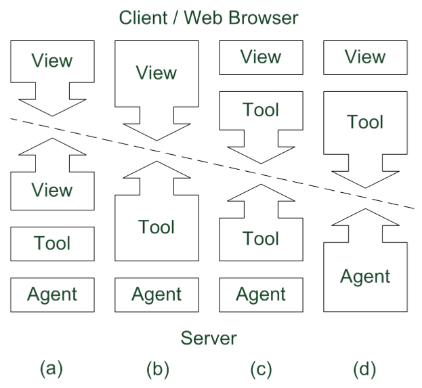


Figure 1: Vertical distribution

b) Splitted View/tool. The vertical cut could also be done between view and tool. In this case the state of all user interface components is completely held on the client whereas the tool has to manage its views from the server. In our case this model has one major disadvantage: As the WOB model suggests inter-tool communication, all the communication between the tools and the resulting interface updates have to be done through the server.

c) Splitted Tool. This disadvantage suggests moving some part of the tool to the client, such that inter-tool communication is already possible on the client. Thus, in possibility c) a part of a tool's logic is implemented on the client. The server-side part of the tool could then take on message passing functionality with the agent system and allows the implementation of server-side logic, like e.g. caches.

d) Splitted Tool/agent. Finally, possibility d) suggests moving nearly all the logic to the client. In this case, the web server only works as a broker between the agent system and the clients. As this solution reduces the network load to a minimum it also burdens the client with all computations and therefore requires a high performance client machine, which is rather unusual in libraries.

In this work case c), the split between the view and the tool, has been chosen because it allows the flexible distribution of tool logic over client and server, but on the other hand also enables a fast inter-tool communication ensuring immediate reactions to user actions and context changes. This reduces latency time and coincidently augments the level of interactivity and enhances effectiveness and efficiency in the usage of the web application.

4.2 Client/Server Communication

The decision to distribute the tool logic over client and server raised the question how to invoke functionality implemented on the server-side from the client and vice versa. Especially the invocation of client-side code from the server, often called "Reverse Ajax" (Mesbah et al., 2006), had to be solved.

The approach we have taken is message driven using the publish/subscribe pattern. Setting up special topics for the client and server side enables the client-side dispatcher to derive from the topic whether the message resides on the client or has to be send to the server using an asynchronous connection. This is also true for the server-side dispatcher which places the message within a client's message queue.

In order to fetch messages from its queue, each client has to poll the queue from time to time using an asynchronous connection. These connections should only be established if they are really necessary. The fact that some of the tool's logic resides on the client makes an efficient polling possible: the decision whether to request messages from the server or not can be accomplished on the basis of the tool's state on the client.

To avoid frequent polling in cases where the client wants to receive a message as fast as possible, we use asynchronous polling as described in (Mesbah et al., 2006). The right choice of the parameters length of a HTTP connection and time between two connections enables us to combine the approaches of synchronous and asynchronous polling. Thus, if fast communication is required we wait with a long connection for new messages in the queue and start a new connection after a short pause, but if e.g. some awareness functionality requires to look only from time to time for new message we employ short connections and a long period of time between two connections.

With this communication model new tools can be integrated easily and developers do not have to be concerned with communication details. After having sent a message to a specific topic the tool just tells the dispatcher that it is waiting for an answer and how fast this answer has to be delivered to the client after being placed in the queue. Messaging to other tool instances running on the same client is also feasible as sending messages to all clients.

4.3 Server-side Architecture

In our model the server hosts both the server-side part of the tools and the application's agent. Due to security reasons we want to parse received messages from the client on the server-side. Together with the aspect that messages in the JavaScript Object Notation format (Crockford, 2006) (JSON) are native data structures on the client and reduce the amount of transferred data in comparison to XML messages we have chosen the JSON format for the client/server communication.

The fact that the XML messages for the agent system are assembled on the server also decouples the agent system from the rest of the web application. If in some point in the future the agents are replaced by web services using e.g. the SOAP protocol or even by some grid services, only the part of the web application dealing with the agent communication has to be changed, leaving the JavaScript implementation intact. Thus, creating another application layer for the agent communication not only decouples this part from the rest of the application but also increases security as well as maintainability.

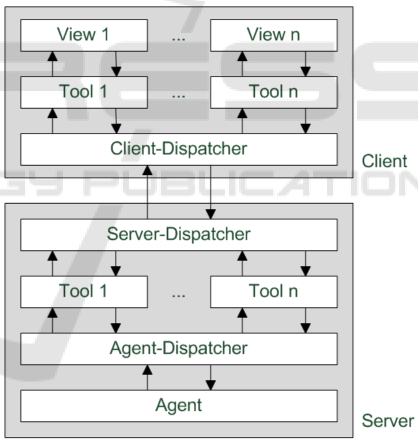
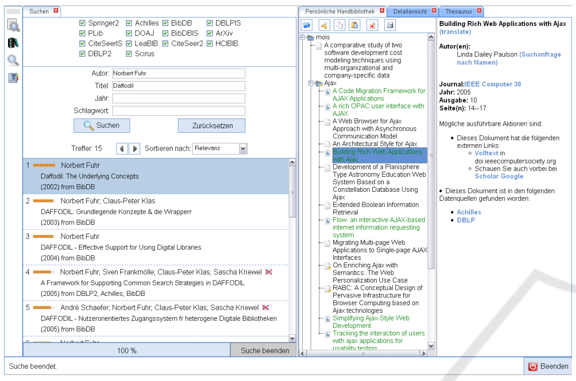


Figure 2: Overview of our tool-based web architecture.

Figure 2 summarizes the resulting tool-based architecture. In order to provide an immediate reaction to user activities the inter-tool communication has been distributed over client and server and therewith provides next to flexibility the possibility to implement a client/server communication based on the tools' states. As some part of each tool still resides on the server an implementation of pro-active functionality is given as it is easy to promote messages to more than just one client. Finally, the Agent-Dispatcher decouples the client/server communication from the agent system allowing further development activities on both sides without affecting the other side.

5 IMPLEMENTATION

In order to evaluate the tool-based architecture described above we implemented a prototype. This prototype runs as a Java web application inside an Apache Tomcat server. We selected the Dojo Toolkit[2] as a JavaScript library providing useful additions to the language standard as well as out-of-the-box user interface components and an inheritance mechanism for object-oriented development on the client-side.



Suchen

☑ Springer2 ☑ Achilles ☑ BibDB ☑ DBLPIS
☑ PLib ☑ DOAJ ☑ BibDBIS ☑ ArXiv
☑ CiteSeerIS ☑ LeaBIB ☑ CiteSeer2 ☑ HCIBIB
☑ DBLP2 ☑ Scirus

Autor: Norbert Fuhr
Titel: Daffodil
Jahr:
Schlagwort:

Suchen | Zurücksetzen

Treffer: 15 Sortieren nach: Relevanz

1 Norbert Fuhr
Daffodil: The Underlying Concepts
(2002) from BibDB

2 Norbert Fuhr; Claus-Peter Klas
DAFFODIL: Grundlegende Konzepte & die Wrapperr
(2003) from BibDB

3 Norbert Fuhr
DAFFODIL - Effective Support for Using Digital Libraries
(2004) from BibDB

4 Norbert Fuhr; Sven Frankmölle; Claus-Peter Klas; Sascha Kriewel
A Framework for Supporting Common Search Strategies in DAFFODIL
(2005) from DBLP2; Achilles; BibDB

5 André Schaefer; Norbert Fuhr; Claus-Peter Klas; Sascha Kriewel
DAFFODIL - Nutzerorientiertes Zugangssystem fr heterogene Digitale Bibliotheken
(2005) from BibDB

100 % | Suche beenden

Suche beendet.

Persönliche Handbibliothek | Detailansicht | Thesaurus

mois
- A comparative study of two software development cost modeling techniques using multi-organizational and company-specific data
- Ajax
  - A Code Migration Framework for AJAX Applications
  - A rich OPAC user interface with AJAX
  - A Web Browser for Ajax Approach with Asynchronous Communication Model
  - An Architectural Style for Ajax
  - Building Rich Web Applications with Ajax
  - Development of a Planisphere Type Astronomy Education Web System Based on a Constellation Database Using Ajax
  - Extended Boolean Information Retrieval
  - Flow: an interactive AJAX-based internet information requesting system
  - Migrating Multi-page Web Applications to Single-page AJAX Interfaces
  - On Enriching Ajax with Semantics: The Web Personalization Use Case
  - RABC: A Conceptual Design of Pervasive Infrastructure for Browser Computing based on Ajax technologies
  - Simplifying Ajax-Style Web Development
  - Tracking the interaction of users with ajax applications for usability testing

Building Rich Web Applications with Ajax (translate)

Autor(en):
Linda Dailey Paulson (Suchanfrage nach Namen)

Journal:IEEE Computer 38
Jahr: 2005
Ausgabe: 10
Seite(n): 14--17

Mögliche ausführbare Aktionen sind:
- Dieses Dokument hat die folgenden externen Links:
  - Volltext in doi.ieeecomputersociety.org
  - Schauen Sie auch vorbei bei Scholar Google
- Dieses Dokument ist in den folgenden Datenquellen gefunden worden:
  - Achilles
  - DBLP

Beenden

Figure 3: Screenshot of the developed web application.

Figure 3 shows our web application's user interface. Next to the toolbar on the left side, providing icons for each tool/view, two tab containers with a resizable bar between them contain the views of the implemented tools. The screenshot shows the view of the search tool on the left as well as the view of the user's personal library. Latter can be used to store documents and search terms for future use. Storing e.g. an entry from the result list can be accomplished by dragging the entry to the library's view. A second view of the search tool providing details for selected search result entries is also available as a thesaurus.

A goal of the implementation was to show that it is possible to implement the WOB model within a modern web browser:

- *Strict Object Orientation and Interpretability of Tools.*
  The architecture allows the encapsulation of strongly related functionality into tools, which are capable of opening views and which are represented through icons in the toolbar (see left side of figure 3). In order to execute functionality, the user can use the implemented drag'n'drop mechanism to drag objects to views or icons.
- *Dynamic Adaptivity.*
  The user interface is dynamically adaptable as it e.g. automatically shows detail information when the user selects an item in the tree (see figure 3), but also hides it when the user starts browsing in order to create more space for the tree and to support the browsing operation.
- *Context Sensitive Permeability.*
  A dedicated object called Desktop, which is shared between all tools, allows the automatic reuse of data in different contexts.
- *Dialog Guidelines.*
  The tools on the client-side are functionally connected as they are able to communicate via the client-side dispatcher. This allows the tools to proactively react to events communicated by other tools. A tool could e.g. open proactively one of its views in order support the user in a given situation.
- *Intelligent Components.*
  With the help of the client-side dispatcher and the Desktop object tools are able to stay informed about their context and can decide to proactively offer their functionality.
- *Status Display with Edit Mode.*
  This software-ergonomical principle reclaims that the state of the application and each of its tools is shown explicitly to the user. Basically this principle can be achieved with the help of dynamic updates of the web page's DOM in an appropriate form. An example in our web application is the status bar showing messages of the tools indicating the tools' states (see figure 3 at the bottom).
- *Iterative Retrieval and Query Transformation.*
  As the transformation of the query results of the contacted digital libraries into a common format is already accomplished by the agent system, the user interface only has to preserve the values entered by the user from query to query in order to simplify the iterative query process. In contrast to web applications of the REST style this is in our case achieved automatically as only parts of the web page are updated when the user submits a query.

6 RELATED WORK

In (Gozali and Kan, 2007) Gozali et al. explore how Ajax can be employed to create a modular OPAC user interface that offers better user experience and task support. By conducting a user survey they conclude that the Ajax pattern has positive effects on usability and functionality. But in contrast to our work, their

[2]http://dojotoolkit.org/

paper focused on the user interface and did not come up with a complete architectural model for the implementation of a modular user interface for Ajax-based web applications.

A web application for geospatial information systems (GIS) requesting images shown to the user from web services is presented in (Cha et al., 2007). As their architecture aims not to be modular nor tool-based requests for the web services in the backend are just routed through the web server. Ajax is used to prefetch images, thus resulting in an improved user response time.

In (Mustacoglu and Fox, 2007) a system for interacting with Collaborative Calendar-Server Web Services from an AJAX-based interface is presented. Their architecture has no intermediary web server as the web browser directly interacts with the web services providing calendaring and scheduling functionality. Opposed to our work the implementation of awareness functionality was not a goal, thus the direct interaction with the web services using SOAP was an adequate solution.

7 SUMMARY & OUTLOOK

This paper presented a software architecture for a web-based user interface for the agent-based and user-oriented digital library system DAFFODIL. Through the implementation of a prototype we could provide some of DAFFODIL's basic functionality over the web. Without the installation of further software users are now able to access DAFFODIL's functionality using a modern web browser. We thereby also transferred the user interface concepts of the WOB model to the web, while at the same time, allowing an easy integration and extension of further functionality. In order to connect the web application to the asynchronously communicating agent system the software model had to be message driven. A component-wise distribution of the views and tools over web browser and server enabled the implementation of a client/server communication which bases on the mediation of the tools' states. Thus, the aim to port DAFFODIL's user interface to the web and therewith showing the applicability of the WOB model and enabling an easy integration of further functionality has been achieved.

The next step is to evaluate the web interface based on the comparison to the evaluation of the Java Swing interface as described in (Klas, 2007). The gained results will be used to further improve the web interface. It is also planned to let users compare our web interface to other user interfaces on the web in order to check the concepts of the WOB model against the concepts of the "Web 2.0".

REFERENCES

Cha, S.-J., Hwang, Y.-Y., Chang, Y.-S., Kim, K.-O., and Lee, K.-C. (2007). Integrating ajax into gis web services for performance enhancement. In *ICCS 2007*, volume 4488/2007, pages 562–568. Springer Berlin / Heidelberg.

Crockford, D. (2006). The application/json media type for javascript object notation (json). http://www.ietf.org/rfc/rfc4627.txt.

Fielding, R. and N., T. R. (2002). Principled design of the modern web architecture. In *ACM Trans. Inter. Tech. (TOIT)*, volume 2, pages 115 – 150.

Fuhr, N., Klas, C.-P., and Gövert, N. (2000). An agent-based architecture for supporting high-level search activities in federated digital libraries. *Proceedings 3rd International Conference of Asian Digital Library*, pages 247–254.

Garrett, J. J. (2005). Ajax: A new approach to web applications. http://www.adaptivepath.com/publications/essays/archives/000385.php.

Gozali, J. P. and Kan, M.-Y. (2007). A rich opac user interface with ajax. In *JCDL '07: Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, pages 329–330, New York, NY, USA. ACM.

Klas, C.-P. (2007). *Strategic Support during the information search process in digital libraries*. PhD thesis, University of Duisburg-Essen, Germany.

Krause, J. (1997). Das wob-modell. *Vages Information Retrieval und graphische Benutzeroberflächen: Beispiel Werkstoffinformation.*, pages 59–88.

Mesbah, A., Broenink, K., and van Deursen, A. (2006). SPIAR: An architectural style for single page Internet applications. Technical Report SEN-R0603, CWI.

Mustacoglu, A. F. and Fox, G. (2007). Ajax integration approach for collaborative calendar-server web services. In *International Conference on Internet Computing*, pages 3–8.

Schaefer, A., Mutschke, P., Fuhr, N., and Klas, C.-P. (2002). Daffodil: An integrated desktop for supporting high-level search activities in federated digital libraries. In *Research and Advanced Technology for Digital Libraries. 6th European Conference, ECDL 2002*, pages 597–612.