Performance-Optimized Adaptation of Personalized Web Fragments.

CONFERENCE PAPER · JANUARY 2003

Source: DBLP

3 AUTHORS, INCLUDING:

Roland Wagner

Johannes Kepler University Linz

223 PUBLICATIONS **1,382** CITATIONS

PERFORMANCE-OPTIMIZED ADAPTATION OF PERSONALIZED WEB FRAGMENTS

Rainer Dlapka, Hildegard Rumetshofer, Roland Wagner
Institute for Applied Knowledge Processing (FAW)
Johannes Kepler University Linz, Austria
{rdlapka, hrumetshofer, rrwagner}@faw.uni-linz.ac.at

Abstract

Current Web users request for individualized and highly-newsworthy information that is immediately delivered by existing Web applications. Therefore, pre-developed, static Web pages which have been regarded as best-practice over years are deprecated to support these demands. Consequently, more sophisticated approaches are needed. One answer to this dilemma is to consider dynamical Web page generation performed on Web Application Servers and being able to respond to individual user requests. The downer for this flexibility is that server-side dynamical Web page generation leads to computation overhead by minimizing response answers. For this, the presented paper will show how to realize performance-optimized individual Web pages. The introduced concept is based on single, cacheable Web fragments.

1. Introduction

The big challenge of adapted user interfaces "is not only to make information available to people at any time, at any place, and in any form, but specifically to say the "right" thing at the "right" time in the "right" way" [1]. It is an alternative to the traditional "one-size-fits-all" methods. Adaptive user interface systems consider individual users having different goals, preferences and knowledge. During interaction with the user, it is capable of adapting to the needs of each single user [2].

Personalized Web applications request for dynamically generated Web pages which are on the fly delivered to individual user requests. It seems obvious that in this context pre-developed, static pages make no sense and are not at all appreciated. However, dynamical Web page generation leads to two big challenges: on the one hand, there is the need for general concepts and procedures for the creation of dynamical responses and on the other hand, high-performant application answers have to be facilitated. An answer to both questions will be given in this paper. The approach presented in this paper has been exemplarily implemented based on Java technology in the DIETORECS project [3].

The paper is organized as follows: Section 2 illustrates related work that considers adaptation aspects, Section 3 describes the proposed Fragment Engine implementing the adaptation concept and process in more detail. Section 4 presents implementation aspects of how to optimize dynamically generated Web applications. Section 5 concludes the paper and describes further research activities.

2. Related work

The DIETORECS project [3] is an initiative of the European Union with its main objective to support tourists with travel recommendations whereby those proposals range from touristic objects such as accommodation, destination or activity to more complex ones, namely, travel packages containing those single items.

The system bases on a set of engines, which are responsible for controlling, steering user requests and responses, delivering status information dependent on certain user or application conditions, extracting personalized recommendations, tracking interesting user data and behavior as well as selecting, adopting and displaying relevant fragments for the requested system status [4]. The work presented here concentrates on the performance-optimized adaptation of the user interface.

3. Fragment Engine

Adaptation in personalized Web applications is threefold: first, content, second, layout and third, navigation [2]. The Fragment Engine is devoted to the adaptation of the user interface concerning layout and navigation aspects. Its capability is to select appropriate Web fragments from a large pool to build individual responses also adjusted to different target devices. Therefore, a fragment describes any chunk of information represented as HTML (hypertext markup language), XML (extensible markup language), WML (wireless markup language) or in case of Java technology JSP (java server pages). In the following, requirements for Web page construction depending on single fragments and the proposed adaptation process are described in more detail.

3.1 Web fragments

Single Web fragments are represented by meta data that is stored in the database. This meta information describes a fragment's kind, physical location, identifier as well layout attributes such as target device or style. Additionally, if a single fragment is embedded into a concrete Web page having specific layout requirements then fragments will be identified through so-called handler.

This concept enables to use one single fragment for different layout representations having different handlers or more-precise several logical instances but only one physical object representation. The physical fragments themselves are currently hosted on a file server, whereby there is the preference to host them also physically in the database. Figure 1 introduces exemplarily the employment of personalized Web fragments.

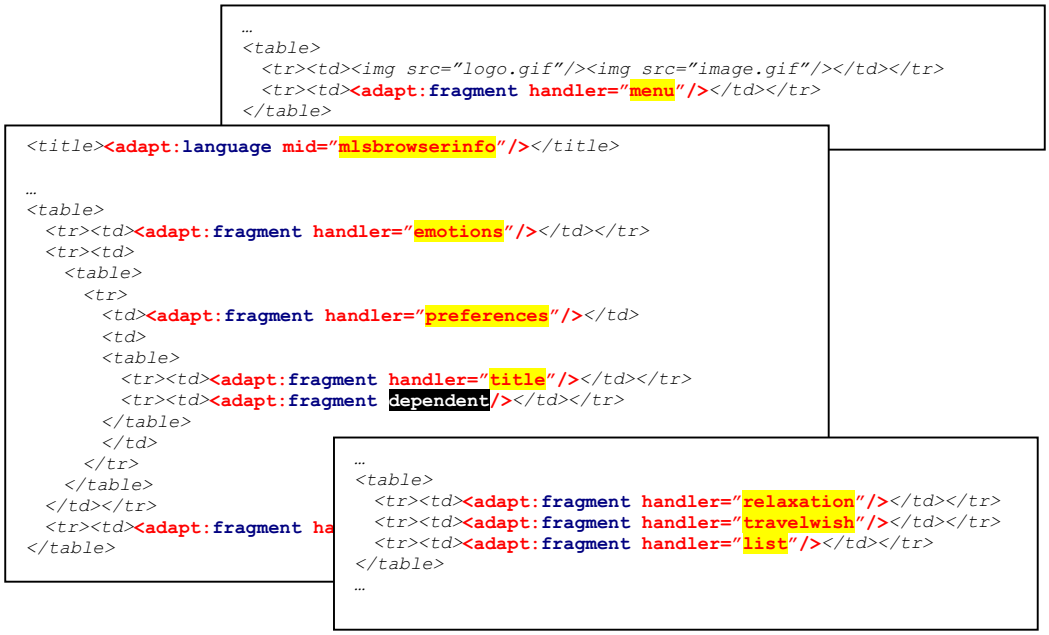

Figure 1. Employment of personalized Web fragments.

Beside, single fragment employment there are two particular cases providing extended flexibility to the presented approach: first, fragment selection depending on a specific workflow status for a certain user and second, multi language support with prevention of fragment redundancy. The current implementation of the Fragment Engine uses therefore three user-defined tags, namely (i) <adapt:fragment handler="…"/>, (ii) <adapt:fragment dependent/> and (iii) <adapt:language mid="…"/> for performing personalized, workflow- and language-dependent adaptation. The adaptation process is dependent on the specific tag.

3.2 Adaptation Workflow

The Fragment Engine is capable of identifying single, workflow-dependent and language-dependent fragments. The adaptation process bases on the identification of single Web fragments according to the proposed tags and the generation of a corresponding Web fragment tree that is interpreted by the engine. In the DIETORECS system, the root of the tree is realized by a certain Java Servlet that handles user requests and responses.

Common fragments, which are at any level in the Web fragment tree are identified through a certain handler. Depending on that handler the engine filters appropriate meta data from the database, selects it on the file server according to its physical location, adapts it with any additional layout requirements and includes the adopted fragment into the calling Web fragment. Figure 2 presents the interface described by single fragments and the resulting Web fragment tree.

Workflow-dependent fragments request for specific consideration. It is assumed that the engine is at any stage informed about user actions and consequently their current workflow status. On the one hand, it has to extract the fragment corresponding to that status and on the other hand, for continuing with common fragment handling it has to

select its specific handler. The information about status – fragment – handler combinations is as well hosted as meta data in the database.

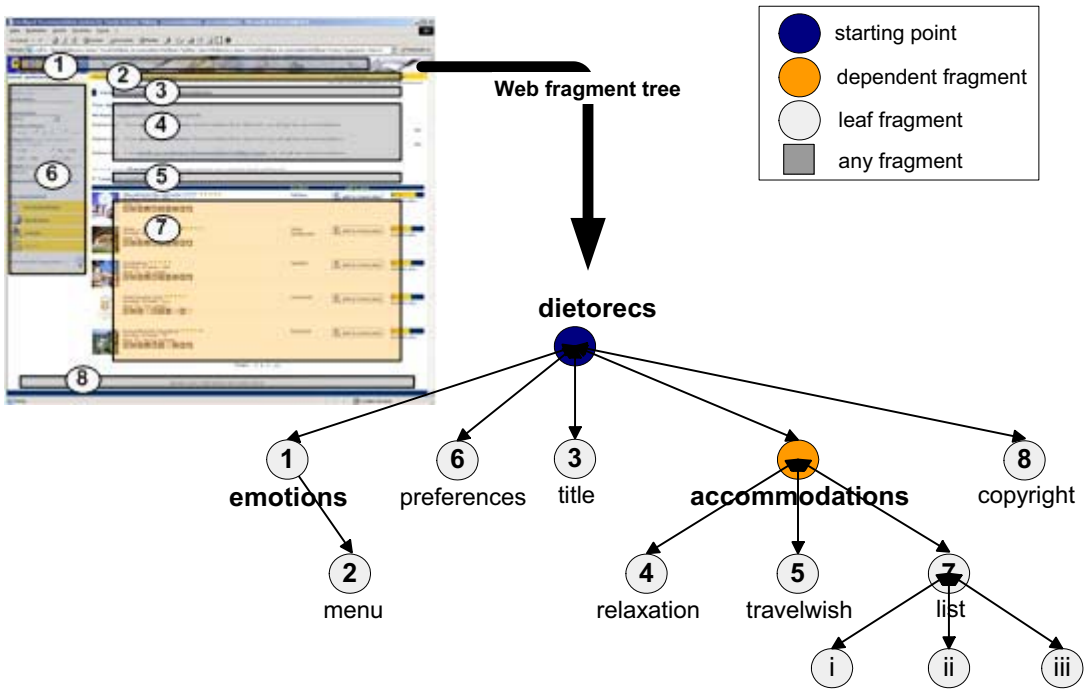

Figure 2. Adaptation Workflow for personalized user response.

Language-dependent fragments are identified via individual keys represented with certain language codes in the database. The selection of a certain language is determined the first time a user opens a user session or in case that he requests any language switches. Depending on the selected language the Fragment Engine is capable of filtering the right text from the database and including it into the calling Web fragment.

So far, the concept for adapting individual Web fragments to personalized Web pages. In the next chapter we will show how to improve performance measure for such pages by using caching capabilities.

4. Performance and Caching

Performance and scalability are critically important for Web applications. Using intelligent caching techniques can substantially increase throughput and response time of a system by reducing overhead. Any Web application that repeatedly uses data can benefit from such techniques.

By caching often used data, the application only needs to get and process data once. Every new request can reuse the cached result that was stored after processing the data first time. Caching within a Web application can be used at several points. Browser and proxy caching reduce the delay for fetching a page and prevent network overload. Server caching can reduce overhead within the generation cycle on the Web

server for a dynamic created page. An intelligent cache manager has to guarantee that the system is informed of every change within the data.

4.1 Basic Principles

The ability to create the content of a Web site at runtime extends the possibilities of Web applications with functions like personalization or interactivity. For example in the DIETORECS system the user is recognized after the login and is greeted by his name. Another big advantage of dynamic content creation is that updates are made on a single place instead of multiple replacing in the static HTML files. Strictly using that ability can separate the content from the presentation.

However using the dynamic scripting paradigm has also disadvantages. The load of the site infrastructure increases since instead of delivering an already built file to the client, every page has to be processed and filled with data before it can be returned. Several steps have to be performed before the creation of the requested page is finished. These steps can create the following delays:

- Accessing persistent storage like databases, file systems
- Network delays due to accessing remote resources like legacy systems, databases or external data feeds
- Data formatting und transformation delays like XML-to-HTML transformation and vice versa
- Interaction bottlenecks like database connection pools
- Overhead of the JVM garbage collection
- Overhead of script execution

These delays are increasing with user load and have a significant negative effect on the site scalability. To lower the effect of these delays several Dynamic Content Caching solutions have emerged. Their common idea is to reuse content that has already been created in a previous processed request. Using this principle prevents the Web server from doing redundant work. It leads to faster response time, higher throughput and lower infrastructure requirements. The Dynamic Content Creation solutions can be grouped into two main types: the Application-specific solutions and the Purpose-built solutions.

4.2 Application-Specific Caching solutions

Application-specific solutions use the support for caching of the application server. So this approach can only be used, if the application server vendor incorporates caching functionality. An example for such a vendor is BEA, which included WebLogic Server Cache tags in the product WebLogic. These functions are based on JSP cache tagging. Page and fragment level caching can be implemented with JSP cache tagging. To achieve fragment-level caching, the script code that should be cached has to be marked or tagged. If the script is processed and a mark is found, the application server will re-use the cached fragment instead of generating it.

The key advantage of the application-specific caching solutions is the fast and therefore cost-effective integration of caching in a Web application. The disadvantage is that JSP cache tags can only be used on presentation layer, which can not reduce

overhead at data layer. Using clustering can also cause problems since multiple caches are required, which might create cache coherency errors.

4.3 Purpose-built Caching solutions

Purpose-built solutions are standalone products that are specifically designed to reduce the disadvantages of dynamic content generation. This solution can be divided into two subtypes: page-level and component-level caching.

4.3.1 Page-Level Caching

Page-level caching solutions cache the output from the application server by intercepting the requests for specific HTML pages. The cache analyses the URL of the request and uses the page name and the request parameters to identify and recognize the specific request next time. It is assumed that the URL has to uniquely identify a page.

The key advantage of this caching strategy is that it can completely offload work from the application server since the whole page is returned from the cache. It also supports personalization as long as the user is identified via a URL parameter. Nevertheless, using the URL as key has also its limitations. There are several situations where a single URL can produce different output, as often used in dynamic Java-based sites. In the DIETORECS system for instance, the business logic determines the content for a specific page depending on a status and condition. Additionally this caching strategy can produce low cache-hit rates if personalization is highly used.

4.3.2 Component-Level Caching

In a component-Level caching solution only components are stored in the cache. Such a component can appear several times within the Web site and even be reused in other sessions. Examples are stock quotes or weather reports. These cacheable components will often create a heavy load on the server if they are not cached since they are expensive to calculate.

In the DIETORECS system some components appear on almost every page, which is optimal for caching (e.g. list items, detail pages). After the first request these components can be stored and reused by every request. The ability of reusing single components within a page is the big advantage of component-level caching. Even if parts of the page change with every request, the server can reuse non changed parts and safe work. Before the implementation of a component-level cache, the components that create performance bottlenecks have to be identified.

This method needs a strategy for invalidating components that became obsolete because their content has changed. These strategies can be categorized into four types: time-based, event-based, observation-based, and on-demand.

A component-level-cache has several advantages. Since most often used components are cached the cache rate is much higher compared to page-level caching. By caching only components instead of whole pages this method is also useful for high personalized sites. The component cache can also reside on a separate machine. This

architecture allows enterprise-class scaling, since the cache can also work with multiple application servers.

A drawback of component-level caching is that components that produce bottlenecks have to be identified. This can be done by load profiling tools or trough site specific knowledge. After this step every component that should be cached has to be tagged. This caching method requires a much higher implementation effort compared to page-level caching.

Figure 3 shows a simple schematic indicating where the two methods fit into a Web site architecture.

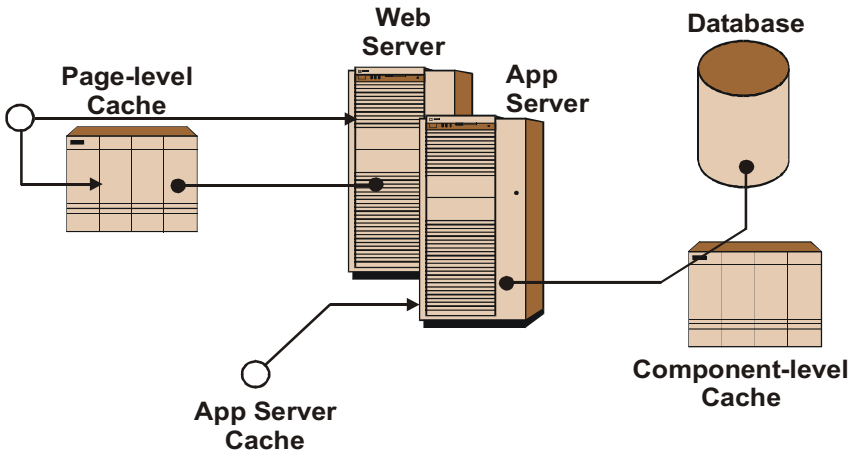

Figure 3. Performance acceleration solutions in Website architecture

“Summing up, Dynamic Content Caching is a valuable technology that should be built into any new Java site’s design and architecture” [5]

4.2 Caching Algorithms

Several algorithms that provide Dynamic Content Caching has been developed. In this chapter we present two algorithms as an example. The following chapter takes a closer look at IBM’s Data Update Propagation Algorithm and shows how we integrate these ideas into our web application

4.2.1. Jakarta Cache Taglib

The Jakarta Cache Taglib is a Java Library based on Java Server Pages 1.1 or higher. This Cache Taglib enables you to cache fragments of JSP pages. With this library you can mark parts of the JSP pages as cacheable. The default caching time for a fragment is 5 minutes and the default cache size is 64k. This values are configurable by the developer. To achieve dynamical caching every fragment has a single key. Fragments with the same key have interchangeable content. For example a fragment whose content depends only on the current user might have ${user} as key. To implement more complex keys for fragments, whose appearance depend on more then value, these values can be concatenated like key=”${user}.${verbosity} to get a single key.

The following example created a fragment that caches a user's birthday [6]:

```
<cache:cache scope="application"
    name="birthdays" key="${user}">
   ... code that retrieves the user's birthday ...
 </cache:cache>
```

4.2.2. FoederCache

The idea of FoederoCache is to push the cache closer to the data source and cache the data that is used instead of the html output [7]. This solution is useful if the bottleneck of the system is the connection to the database and not the generation of the dynamic web pages.

By using this cache the number of calls to the data source is reduced. This caching algorithm also minimize the overhead that is created by triggers and event handlers, that are necessary to respond to changes if fragment caching is used.

4.3 Data Update Propagation

The problem with the presented methods for dynamic content caching is that data changes are not immediately updated at the presentation layer. A news ticker component, for instance, often changes and this change has to be done without any delay. Consequently, a method is needed that synchronizes the cache with the database. Every cached object has to be identified with the data that it represents. IBM has developed the so-called Data Update Propagation algorithm (DUP) [8] to guarantee that every component will be updated if the data has changed.

The basic approach of DUP is to create a connection between the object that will be cached, and the data that it represents. Since the cache only stores HTML code, object and data are disjoint. The storage for the cache is managed by a cache manager. The cache manager manages data dependency information between objects and the data that they represent. In case that the cache manager retrieves information about a change in the data, it has to identify every object effected through the dependency information. These objects get invalidated because of the change of its underlying data.

The application program has to determine which data is represented by which object and create a join. For example this join can be created by the ud_id (underlying data ID) that is used in the database to identify the data. The cache manager stores the information about which object represents which ud_id. The application program is responsible to pass this dependency information to the cache manager. This can be done via an API function [6]:

```
add_dependency(obj_id ,ud_id)
```

In case that data changes in the database the cache manager has to be informed via another API function:

```
underlying_data_has_changed(ud_id)
```

The cache manager invalidates all objects that are joint to *ud_id*.

DUP was developed by IBM Research in 1998 for the Olympic Games Web site. As a result of DUP and pre-fetching IBM was able to achieve cache hit rates close to 100% throughout the entire Olympic Games [9]. This high hit rate allowed the system to serve pages quickly even during times of peak demand. The site contained approximately 87,000 unique pages of which approximately 21,000 were dynamically generated.

4.4 Caching Model for the DIETORECS Fragment Engine

The caching method used in DIETORECS is based on the Data Update Propagation Algorithm. The cache manager stores the mapping of fragments and data. There are four different data objects in the database for DIETORECS:

- Accomodation Objects (acc_id)
- Destination Objects (dest_id)
- Interest Objects (int_id)
- Multilanguage String Objects (string_id, language_id)

The first three object types are identified by a unique id. Multilanguage string objects are identified by an id and the language id. There are two different add_dependency methods to map a fragment to data:

```
add_dependency(fragment_id ,ud_id)
add_dependency(fragment_id, language_id ,mso_id)
```

If the underlying data in the database is changed the application will have to inform the cache manager to invalidate the affected fragments.

```
underlying_data_has_changed(ud_id)
underlying_data_has_changed(language_id ,mso_id)
```

Each fragment that can be cached is marked by a tag to inform the cache manager about the data that it presents. For example the fragment FRAACCDETAIL shows details of an accommodation. If this fragment is part of the page that is requested by a client it will inform the cache manager about its fragment id and the accommodation id. Each time the fragment for this accommodation is requested the cache manager will bypass the creation of the fragment and take the HTML from the cache instead.

After a Web server restart the cache manager is initialized and contains no mappings. The first page request causes that all fragments for this page are generated. Fragments that are tagged to be cached are stored in the cache. With every new request the cache is filled. If a data object (e.g. an accommodation) is altered by using the Web interface, the application will brief the cache manager that the object identified by obj_id has changed. The cache manager invalidates all fragments in its cache that signalized to present data of this object. If such an invalidated fragment is requested next time a new instance will be created and stored, instead of bypassing and reusing the object from the cache.

The goal of using this caching model in the DIETORECS system is to accelerate especially detail pages (e.g. for accommodations or destinations) and result lists, which have been identified as bottlenecks in our performance tests. It is also possible

to cache static fragments such as the left side menu and the header by calling the add_dependency method without an ud_id, which gives additional speed up even for each single page.

5. Conclusion

In this paper we have presented an approach for performance-optimized dynamically generated Web pages. First, we introduced the Fragment engine which is devoted to the adaptation of the user interface concerning layout and navigation aspects in personalized Web applications, and second, we discussed several problems that can appear when using high personalized and therefore dynamical pages. Finally, we presented the Data Update Propagation algorithm that was developed by IBM for the Olympic Games Web site and we described the adaptation of this algorithm to accelerate the generation process of dynamic pages in the DIETORECS system.

As the DIETORECS system is still under development improvements concerning Fragment engine and optimization aspects will be deepened and continued.

6. Acknowledgement

This work has been partially funded by the European Union's Fifth RTD Framework Program (under contracts DIETORECS IST-2000-29474).

References

[1] Fischer, G., User Modeling in Human-Computer Interaction, User Modeling and User-Adapted Interaction 11 (1-2): pp 65-86, (2001).
[2] Brusilovsky, P., Adaptive Hypermedia, User Modeling and User-Adapted Interaction 11: pp 87-110, (2001).
[3] Homepage of the IST-2000-29474 DIETORECS project: Intelligent Recommendation for Tourist Destination Decision Making, http://dietorecs.itc.it/ (2002).
[4] D. R. Fesenmaier, F. Ricci, E. Schaumlechner, K. Wöber, and C. Zanella, "DIETORECS: Travel Advisory for Multiple Decision Styles", in Proceedings of Enter conference, Helsinki, Finland, January 29 - 31 , 2003.
[5] Thomas, H., Accelerating Java Web Application Environments with Dynamic Content Caching, Java Developers Journal.com: pp 34-38 (July 2001).
[6] The Jakarta Projekt: Cache Taglib, http://jakarta.apache.org/taglibs/doc/cache-doc/intro.html.
[7] Foedero Technologies Inc. : A Technical White Paper by Foedero Technologies Inc. http://www.db2link.com/Papers/Papers_full_listing/FoederoCache%20White%20Paper.htm ,Oct. 2001
[8] Arun Iyengar, Jim Challenger, Data Update Propagation: A method for Determining How Changes to Underlying Data Affect Cached Objects on the Web, IBM Research, (1998).
[9] Arun Iyengar, Jim Challenger, Paul Dantzig, A Scalable System for Consistently Caching Dynamic Web Data, IEEE INFOCOM '99 Proceedings: The Future Is Now. , New York, IEEE, pp 294-303.