Distribution and Synchronization of Context Modeling Mechanisms between Servers and Clients on the Web

Michael Hinz *Zoltán Fiala*

Dresden University of Technology, Department of Computer Science
Heinz-Nixdorf Endowed Chair for Multimedia Technology
D-01062, Germany, +49-351-463-38516

{michael.hinz, zoltan.fiala}@inf.tu-dresden.de

***Abstract*—The on-the-fly generation of personalized context aware Web applications is very time consuming and causes an enormous server load. For this reason, such applications are still restricted to specific application domains. To meet this challenge, this paper proposes a novel component-based approach for dynamically distributing server load to clients. Server tasks like context modeling algorithms are transferred and carried out on end devices. A distribution manager was developed to decide whether a task can be performed on the client. This is done by monitoring components observing current client capabilities and system states of the Web server. Furthermore, mechanisms for synchronizing context models between servers and clients are provided. Proving feasibility the paper demonstrates the distribution of a user modeling mechanism based on a prototype.**

Key words: distribution, synchronizing, context awareness, adaptive architectures

1. INTRODUCTION

Today more and more (mobile) devices with heterogeneous capabilities are getting access to the WWW. Other trends like Location Based Services and personalization of Web contents require adaptation mechanisms taking various context data into account. Therefore, modern ubiquitous Web systems have to deal with varying context information in order to support context awareness. Accomplishing this requirement necessitates gathering, processing and representing context information, so that it can be used for the adaptation. Web pages have to be generated dynamically according to the context that can change during each Web page request. Assuming many users performing requests simultaneously, the modeling of context and the reiterative online generation produces an enormous server load. Reducing this server load is one of the key factors for the commercial success of context aware systems.

For that purpose several strategies (mostly caching mechanisms) exist which can be divided into a few main categories. Some of the approaches concentrate on the caching of dynamically generated content [1] on the server side ([2], [3]). Still, dynamically generated adaptive Web pages are highly dependent on the context in order to support personalization, device independence and context awareness. Therefore, the hit ratio of those caches is very low [1].

Other strategies focus on caching dynamic content on proxies. Interesting approaches are proposed by [4] and [5] which cache dynamic/active content by migrating the content generating scripts and the data used by them from the server to proxies close to the clients. However, though the latency caused by the backbone delay could be reduced, the hit ratio problem was not solved but only shifted to the proxy server.

Other methods for reducing server load distribute user requests to different servers by using load balancing and clustering mechanisms [6] and [7]. Furthermore, there has been done some work in the area of adaptable software architectures [8].

In this paper we propose a distribution mechanism that can be effectively used to reduce server load by shifting system components (e.g. modeling components and content generation components) of a context aware Web architecture to the client. Furthermore, a mechanism for the synchronization of the context data between the server and the client is presented. Thus we provide an adaptable context aware Web system architecture that dynamically adjusts itself to changing server load.

2. CONTEXT AWARE WEB ARCHITECTURES

Context aware Web systems dealing with different context information have a more complex architecture than conventionally Web systems. On the basis of the AMACONT system architecture [9] Figure 1 shows the general structure of such an architecture, consisting of several components for modeling context information and representing it in a context model. According to this context model document generation components adapt the requested Web document. Thus, personalized and location aware Web applications can be generated for the ubiquitous Web.

In this paper we want to show how complex modeling tasks and parts of the dynamic document generation can be transferred to the client side in order to reduce server load. As an example we choose a personalization mechanism. Aim of this mechanism is to allow users to interact with media components contained in a Web page, to automatically derive user preferences from those interactions and to dynamically update the resulting Web presentation according to these preferences. For this purpose user interactions are tracked on the client device and sent to the server. On the server a user modeling component calculates user preference rules and updates the user profile of the context model (see Figure 1, right components in the boxes "Context Modeling" and "Context Model"). Based on that profile the requested Web documents can be adapted, respectively (Figure 1, middle component of the document generation). To guarantee the best adaptation these processes are performed whenever the user is requesting a new Web page.

In general this is not only a problem of this specific user modeling mechanism but of all context aware systems. In order to achieve the best QoS properties the generated Web documents have to be recalculated whenever the context (e.g. location of the user, device properties, environmental properties) changes. This causes server load and reduces the overall system performance and therefore the response time of the server.

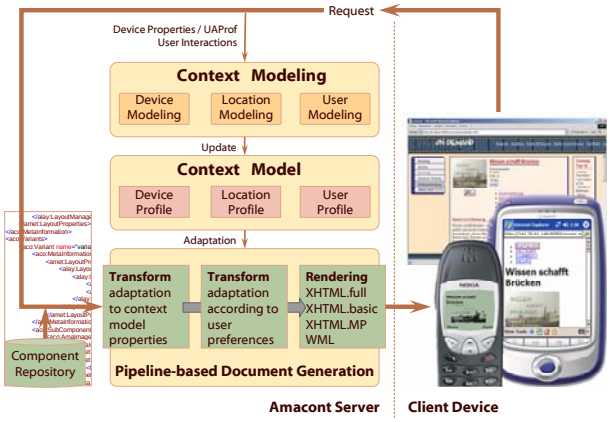

Figure 1: General Structure of a Context Aware Web System Architecture

3. ENHANCING PERFORMANCE BY DISTRIBUTING AND OMITING SERVER TASKS

Figure 1 shows that the general structure of a context aware Web system architecture consists of a set of different components (e.g. modeling and transformation components). Some of them like the rendering to the concrete output format have to be processed each time the user requests a document. Others (e.g. user modeling) that are only improving the quality of the generated document/presentation are optional.

In order to reduce server load one possibility is to process optional tasks only if there are enough server resources available. Another solution is the establishment of a distribution mechanism that shifts components to the client device if it has the capabilities to process those components. Both mechanisms can be effectively used to reduce server load (see Section 4). For that purpose different requirements have to be considered:

For enabling a decision if there are enough server resources available or if the client has the capabilities to execute a component, states of the overall system have to be monitored. Moreover, descriptions of the components' requirements (Which resources are needed? Are there alternative implementations of the component?) and their necessities (optional or mandatory?) are needed. Furthermore, rules for the distribution or omitting of components have to be specified. Based on that specification the system can react on changing system states by adapting its structure/architecture.

To put those requirements into practice, Figure 1 shows again the general structure of a context aware Web system extended by a distribution mechanism. This mechanism enables to monitor states of the server and to acquire device capabilities. According to this information distribution actions are triggered that shift server load to the client device. Note that the shifting of server components can only be done when all underlying data is available on the client side.

As an example Figure 2 (see the dotted areas) illustrates the distribution of the user modeling mechanism. This means that the components "User Modeling", "User Profile" and the corresponding transformer can be performed either on the server or on a capable client device.

3.1. Monitoring System States

As already mentioned, monitoring the state of the system is a key factor for performing distribution mechanisms. If components should distribute to the client device, monitoring only states of the server is not sufficient, i.e. the client device's capabilities have also to be captured in order to detect whether it has the resources to execute the components.

In Figure 2 a monitoring manager is shown that provides a plug-in mechanism for different monitoring components. Furthermore, it offers a common interface for retrieving system states to the distribution manager component (see Section 3.2). The next sections (3.1.1 and 3.1.2) describe the functionality of two monitoring components by an example.

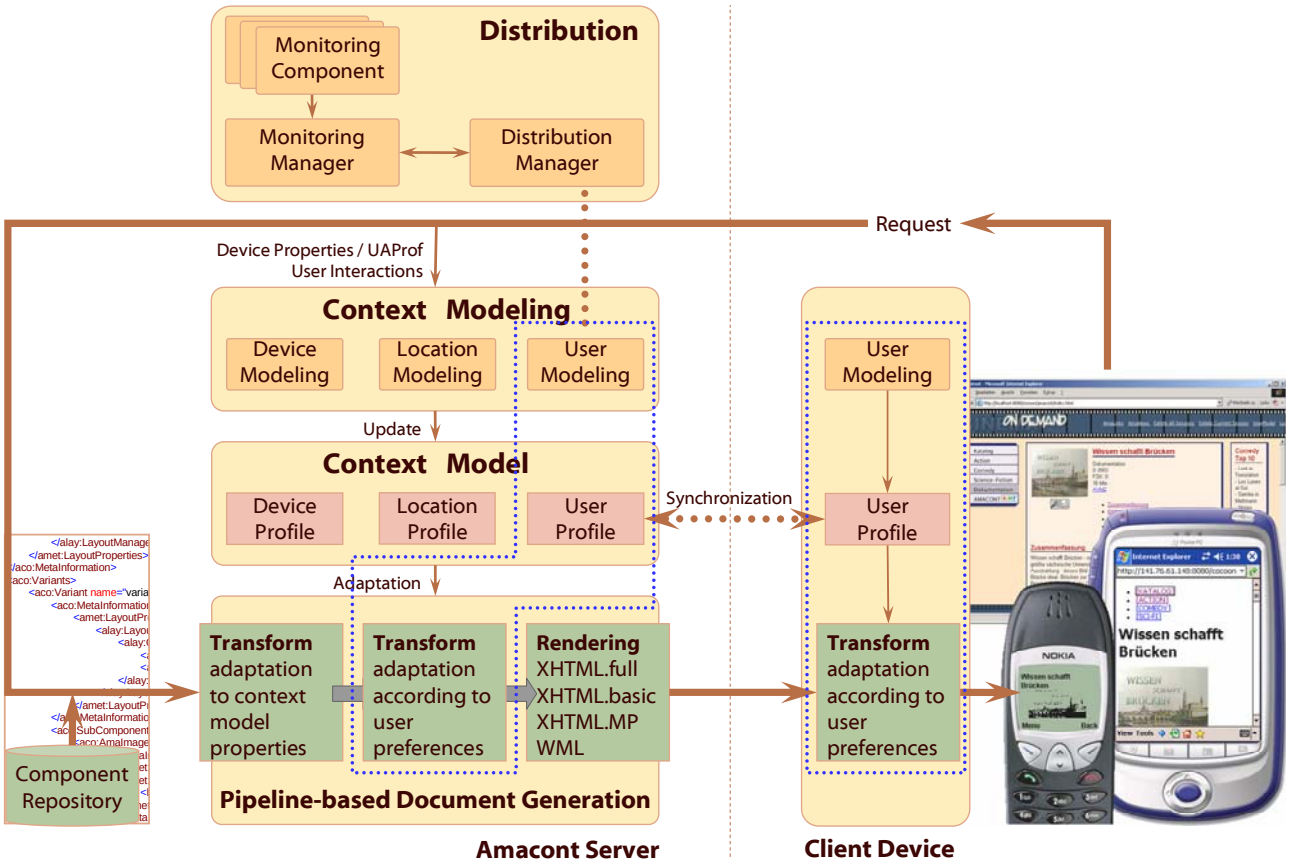

Figure 2: Extended System Architecture Supporting Distribution of Server Tasks (Example: User Modeling)

3.1.1. Measuring System Server Load

In order to find out whether server tasks should be omitted or executed on the client, mechanisms for monitoring the server load are required. This is e.g. possible by a monitoring component that measures the processor time consumption of the server system. Due to the high variability of the current time consumption this has to be done for a period of time to assure reliable values. Another possibility is to count the number of server requests during a specific time period. This concept does not give a direct statement about the server load but gives an indicator that can be mapped to former behavior patterns.

3.1.2. Measuring Device Capabilities

To acquire device capabilities several strategies exist. The most popular method is to analyze the HTTP user-agent parameter that comes with the HTTP request and map this parameter to a device or browser repository on the server side. However, this works only for a few nearly static device properties. The usage of the User Agent Profile specification (UAProf [10]) which is based on the CC/PP framework [11] establishes a more effective mechanism for gathering dynamically changing device properties on the server by analyzing UAProf enabled requests. Unfortunately, this specification only provides a common vocabulary for WAP devices. Still, most of the vocabulary can be adopted for other non WAP devices like e.g. Web browsers on desktop computers, notebooks or PDAs. In our work we extended this vocabulary in order to support those device classes. Furthermore, for these device classes we also provide a mechanism to transfer the gathered device capabilities within the HTTP request to the server [12].

In this way our device modeling mechanism illustrated in Figure 3 distinguishes between UAProf enabled devices, devices providing the user agent and devices giving support for client side code fragments like JavaScript, Jscript and Java (combinations are possible). Such client side code fragments are included during the Web document generation on the server and directly gather device properties on the client [12]. The gathered information is encoded in a UAProf like representation and integrated in the HTTP request by a client/server communication component for processing that information on the server.

According to the gathered capabilities from the client, the server processes the corresponding device context. The processed context is represented as the device profile in the context model (see Figure 1 and Figure 3). The representation is based on the above mentioned

extended UAProf format. The processing of the device context on the server depends on the obtained request.

1. If the request only includes the user-agent parameter, this parameter is mapped to the according device profile in a device repository. Note that by using only this mechanism dynamically changing device properties (e.g. bandwidth or size of the browsers window) can not be taken into account.
2. If a UAProf enabled device sends a user-agent profile or a difference profile within the request, that information is handled by DELI [13] on the server side which provides an API for Java servlets to determine client capabilities using CC/PP and UAProf. The output of the DELI component makes a profile representing UAProf properties available.
3. Whereas today nearly only WAP 2.0 devices support UAProf, our system is also able to autonomously collect the devices properties of other end devices (e.g. Notebook, PDA) via the above mentioned client side code. The on the client gathered properties are sent within the HTTP request. Our server processes that information and merges it with an existing or by DELI generated device profile.

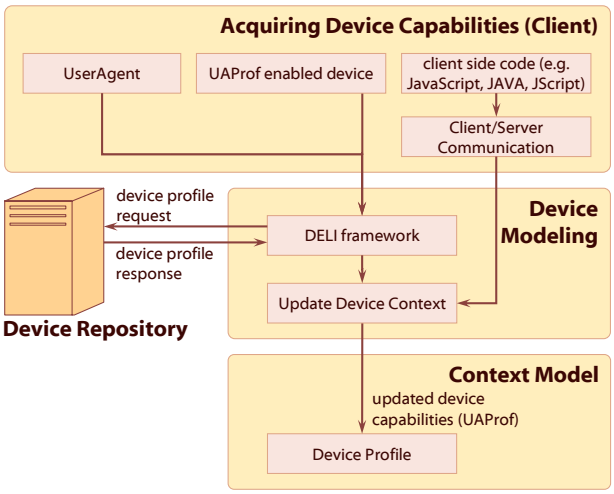

Figure 3: Modeling Device Capabilities

The usage of these mechanisms [14] enables to acquire even permanently changing device properties. The result is an always up-to-date device profile of the context model. With our implemented monitoring component the states and capabilities of the current client device can be effectively used for distribution strategies.

3.2. Defining Distribution

Having the knowledge about various system states, the distribution and/or omitting of system components can be triggered. This is done based on a distribution logic according to which the system can react on changing system states by adapting its structure/architecture. The following code fragment shows a distribution logic that can be attached to a component of the system architecture. This example is taken from the system component that handles the client side execution of the user modeling component and the according adaptation transformation:

```
<Logic junctor="and">
  <MonitorComponent name="ServerPerformance">
    <Parameter name="cpu_usage"
               comparator=">" value="60"/>
  </MonitorComponent>
  <MonitorComponent name="ContextModel">
    <Parameter name="/DeviceProf/Browser/JavaEnabled"
               comparator="==" value="true"/>
  </MonitorComponent>
</Logic>
```

The code shows that the component and the transformation are only carried out if the processor's time consumption (measured by the "ServerPerformance" monitoring component) exceeds 60 percent and the client device supports the execution of Java applets (measured by the "ContextModel" monitoring component that evaluates the device profile of the context model). The distribution logic of the system component that handles the server side user modeling mechanism looks similarly. In that case the components are executed only if the processor's time consumption falls below 60 percent or if the client device does not support Java applets.

3.3. Synchronizing Context Information

With the distribution mechanisms presented in the last sections the processing of server side components can be omitted and the components that are executable on the client are integrated into the Web pages during their generation. Still, the client side execution of components requires not only the components themselves but also the underling data. In our user modeling example this means that also the user profile containing the rule based representation of user preferences is needed in order to control the transformation.

For that purpose we designed and implemented a mechanism that synchronizes the client and the server side context models (Figure 2). To provide versatility, this mechanism does not affect the normal Web application and its delivery to the client. The context data is automatically included (by using hidden forms or applet initialization parameters) into the generated Web pages during the rendering process. The changed context data is communicated back to the server within an HTTP request. Therefore, the same mechanism as the one for sending the automatically gathered device capabilities is used (see Section 3.1.2).

Those synchronization mechanisms of course produce additional network traffic that raises latency. Still, in the most scenarios the synchronization has to be performed only when a user enters (login) or leaves (logout) the Web application. Other scenarios in which server side

components and client side components depend on the same data structures are possible but should be avoided in order to keep the latencies short.

4. EVALUATION

A short evaluation of the proposed distribution mechanism can be seen in Figure 4. The diagram shows the server response time according to the number of interactions the user has done with a Web application. The first (blue) curve represents the status before the distribution mechanism is launched and shows a continuous growth according to the number of interactions done by the user. This is because for every interaction a set of rules representing the user's preferences has to be updated or extended during user modeling. The second (pink) curve shows the situation when the distribution manager decides to perform the user modeling component on the server side and is therefore also continuously growing. Since the distribution management requires some additional time, the response times are even a little bit longer (average: 8,6ms). This time is also needed by performing the modeling component on the client side. The third (yellow) curve indicates the client-side execution of the user modeling component and shows that the number of interactions has no effect on the server response time. This means that depending on the number of interactions the distribution of the user modeling reduces the server load and thus the response time (10% at 3-4 interactions).

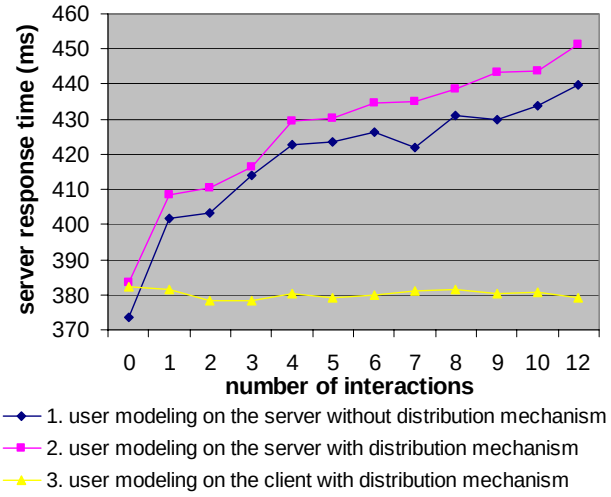

Figure 4: Performance Measurements

5. CONCLUSION AND FUTURE WORK

In this paper we proposed a component-based approach for reducing server load by dynamically omitting optional server tasks or executing them on the client devices. It was shown how system states can be measured with monitoring components in order to use that information for distribution mechanisms that are handled by a distribution manager. Furthermore, methods for synchronizing the underlying data structures (e.g. context information) between the server and the client were presented. Finally, performance improvements were evaluated by specific measurements. By using our distribution logic (see 3.2) the execution of system components can be efficiently triggered. Still, even though this mechanism enables to define elaborated distribution strategies, the definition of complex system adaptation processes can be a difficult task for the system administrator. In the near future we want to decrease this complexity by visual administration and configuration tools. Further future work will concentrate on minimizing the server load by additional fragment-based cache strategies aiming to reduce the size of the documents that were processed during the pipeline-based document generation.

REFERENCES

[1] Barish, G., Obraczka, K., "World Wide Web Caching: Trends and Techniques", *IEEE Communications, Internet Technology Series*, May 2000.

[2] Zhu, H., Yang, T., "Class-based cache management for dynamic web contents", in *Proceedings of IEEE INFOCOM*, 2001.

[3] Iyengar, A., Challenger, J., "Improving web server performance by caching dynamic data", in *USENIX Symposium on Internet Technologies and Systems*, 1997.

[4] Cao, P., Zhang, J., Beach, K., "Active cache: Caching dynamic contents on the web", in *Proceedings of the IFIP Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, 1998.

[5] Calo, S., Verma, D., "An architecture for acceleration of large scale distributed web applications", 2002.

[6] Huang, C., Sebastine, S., and Abdelzaher, T., "An Architecture for Real-Time Active Content Distribution", *In Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS 04)*, 2004.

[7] Bourke, T., "*Server Load Balancing*", O'Reilly & Associates, ISBN: 0-596-00050-2, August 2001.

[8] Comquad Project Homepage: http://www.comquad.org/

[9] Hinz, M., Fiala, Z., "AMACONT: A System Architecture for Adaptive Multimedia Web Applications", *(XSW 2004), Berliner XML Tage*, Berlin, October 2004.

[10] Wireless Application Group, "User Agent Profile Specification", *Open Mobile Alliance WAP Forum*, 2001.

[11] Klyne, G., Reynolds, F., Woodrow, C., Ohto, H., Hjelm, J., Butler, M., and Tran, L., "Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0", *W3C Recommendation*, January 2004.

[12] Hinz, M., Fiala, Z., "Personalization-based Optimization of Web Interfaces for Mobile Devices", in *Proceedings of the MobileHCI 2004*, September 2004, Scotland.

[13] Butler, M, "DELI: A DElivery context LIbrary for CC/PP and UAProf", *HP, External Technical Report HPL-2001-260* (revised version 02/08/2002), 2002.

[14] Hinz, M., Fiala, Z., "Context Modeling for Device- and Location-Aware Mobile Web Applications", *3rd International Conference on Pervasive Computing (Pervasive 2005), PERMID 2005*, München, May 2005.