Transparent Multi-Client Access and Dynamic Content Generation

Hrvoje Komerički, Damjan Lončarević and Tomislav Fitz
Department of Telecommunications
Faculty of Electrical Engineering
University of Zagreb
Unska 3, HR-10000 Zagreb, Croatia
{hrvoje.komericki, damjan.loncarevic, tomislav.fitz}@fer.hr

Abstract—**The proliferation of mobile devices has created the need for delivery of content tailored to capabilities of the device used. This content adaptation has to be invisible to the end user. Our goal was to find a solution that would enable this adaptation for a hierarchical data structure that is changing very frequently. This means the content has to be created dynamically. In this paper we propose a solution for dynamic content generation and transparent multi-client access. Clients send requests that contain parameters, and headers that describe them and the content they want to receive. The solution for content generation comprises: a request handler that processes the request and parameters so that content based on them can be generated, a data service that delivers the data needed for content generation, a content transformer that transforms provided data to the format that is expected by the client in the response, and a response generator that generates a response that contains the requested content. Transparent access is provided using the User-Agent header field sent in the client request. We explain the architecture of solution and the general content generation process. Implementation of the proposed solution is demonstrated in a case study involving visualization of GRID monitoring data.**

I. Introduction

There has been a lot of recent work directed towards presenting content on mobile and hand-held devices. A software architecture to support computing on hand-held devices has been proposed in [1]. An overview of current technologies and standardization efforts for enabling device-independent Web applications may be found in [2]. One of the main technologies for adapting standard Hypertext Markup Language (HTML) content is transcoding [3], which unfortunately has known limitations when it comes to more complex structures. A recent approach to transcoding Web pages for display on mobile devices attempts to take into account semantics as well [4].

In this paper we present a solution for transparent access to dynamically generated content. It was our goal to make possible for end users to access the content using different devices they request the same way, independent of the device they are using. The content is generated from data that are extracted from a database. These data are changing frequently so content cannot be generated statically as in previous solutions. We had to find a solution that would enable dynamic content generation. First Java Server Pages (JSP) seemed to be a good solution for dynamic content generation, but we found it more elegant and faster to combine content generation with transparent access. Our solution uses Apache Cocoon Servlet Technology, an Extensible Markup Language (XML) [6] publishing framework that uses XML and Extensible Stylesheet Language Transformations (XSLT) [7] technologies for server applications, as described in Section II. The description of the content generation process and general architecture of our solution is described in Section III. Our implementation, which is described in Section IV, enables devices with different browsers to access content that is adapted to their browser type. The data is received from a data server in the form of an XML document and then browser-adapted content is generated using XSLT. Our implementation was tested using the Sony Ericsson P800 phone, the iPAQ pocket PC devices and Deckit 1.2.3 WAP emulator with the data provided by MonALISA system [8], a distributed monitoring system used in this case to monitor AliEn Grid sites (http://alien.cern.ch/). The results are described in Section V. In Section VI the conclusion and the directions for future work are given.

II. Technology

We used Apache Cocoon Servlet Technology (http://cocoon.apache.org), an XML publishing framework that uses XML and XSLT technologies for server applications. XML is a very flexible markup language designed to describe data structure and semantics. An XSLT stylesheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an XML document that uses a formatting vocabulary, such as HTML or WML. Cocoon is designed for performance and scalability around pipelined SAX processing and offers a flexible environment based on a separation of concerns between content, logic, and style. File transformation in Cocoon is based upon a pipeline architecture as shown Figure 1.

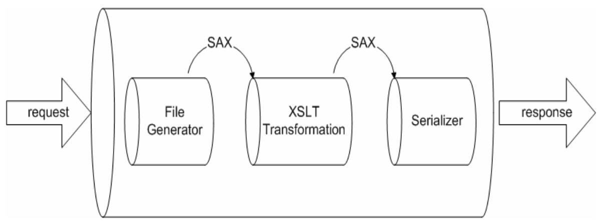


Figure 1. Cocoon pipeline

An XML document is pushed through a pipeline. Every pipeline begins with a generator, continues with zero or more transformers, and ends with a serializer. We will explain the components of the pipeline in more detail.

The Generator is the starting point for the pipeline. It is responsible for delivering SAX events down the pipeline. The simplest Generator is the FileGenerator: it takes a local XML document, parses it, and sends the SAX events down the pipeline. The Generator is constructed to be independent of the concept "file".

A Transformer can be compared to an XSL: it gets an XML document (or SAX events), and generates another XML document (or SAX events). The simplest Transformer is the XalanTransformer: it applies an XSL to the SAX events it receives.

A Serializer is responsible for transforming SAX events to a presentation format. For actors looking at the back of the pipeline, it looks like a static file is delivered. A browser can receive HTML, and will not be able to tell the difference from a static file on the file system of the server. There are Serializers for generating HTML, XML, PDF, VRML, X3D and WAP. The simplest Serializer is the XMLSerializer: it receives the SAX events from up the pipeline, and returns a "human-readable" XML file.

Besides using the various components, we can use matchers, and selectors to choose a specific pipeline processing. The pipelines are defined in Sitemap. It contains configuration information for a Cocoon engine: list of matchers, generators, transformers, readers, serializers, selectors and processing pipelines with match patterns. Sitemap is an XML file corresponding to a sitemap Data Type Description (DTD). It can be edited to add new elements. Sitemap is generated into a program and is compiled into an executable unit.

III. SOLUTION

Figure 2 shows the basic physical architecture of the system. Clients access the server through the network and request content that is generated dynamically at the origin server. Data that is used for content generation is extracted (from a database) by another server and is sent to the origin server in a common data format. Generated content is adjusted to parameters and headers that are sent in the client's requests and is sent back to the clients. It is important that the users from different clients can access the content transparently. This means that the user requests content in the same way no matter what kind of device he/she uses and has no knowledge that the content is dynamically generated and adjusted to parameters sent by the client.

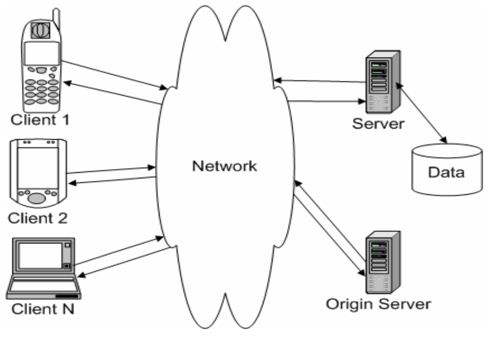


Figure 2. System architecture

The content generation process is shown in Figure 3. The client sends the request, which contains parameters, and headers that describe it and the content it wants to receive. The Request Handler processes the request and parameters so that content matching them can be generated. Data for content generation is requested from the Data Service. It has the task of extracting and adjusting data (from a database) according to the sent parameters. Also it formats the data to the format used by the Content Transformer.

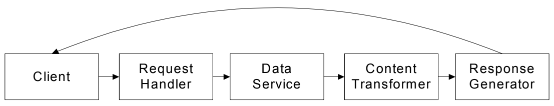
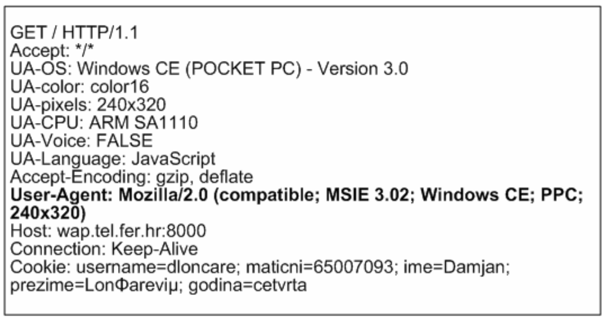


Figure 3. Content generation process

The provided data is then transformed in the Content Transformer to the format that is expected by the client in the response. The Response Generator generates the response that contains the requested content and it is sent back to the client. The client receives the generated content and it can be displayed to the user. The architecture described in Figure 3 is logical. Usually the Request Handler, the Content Generator, and the response Generator are physically located on the origin server. The Data Service is usually located on another server that has access to a database.

IV. IMPLEMENTATION

Users can access the content from different devices. User agents (browsers) from different devices request the content from the server sending an HTTP request with the User-Agent header field set as shown in Figure 4. Cocoon identifies the type of browser through the value of the User-Agent header field. The browser type is used to select the specific transformer for the used browser in the pipeline. After serialization a response is sent back to the client.


```
GET / HTTP/1.1
Accept: */*
UA-OS: Windows CE (POCKET PC) - Version 3.0
UA-color: color16
UA-pixels: 240x320
UA-CPU: ARM SA1110
UA-Voice: FALSE
UA-Language: JavaScript
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/2.0 (compatible; MSIE 3.02; Windows CE; PPC;
240x320)
Host: wap.tel.fer.hr:8000
Connection: Keep-Alive
Cookie: username=dloncare; maticni=65007093; ime=Damjan;
prezime=LonΦareviµ; godina=cetvrta
```
Figure 4. HTTP header

The initial process of user identification may be extended in the future with the client profile specified based on Composite Capabilities / Preference Profile (CC/PP) proposed by W3C (http://www.w3.org/TR/CCPP-struct-vocab/) [9], or by using Session Initiation Protocol (SIP) extended with Session Description Protocol (SDP). CC/PP is a client profile data format used for describing device capabilities and user preferences based on the Resource Description Framework (RDF).

The server used in our implementation was Intel Pentium 4 (1.6GHz). It had Apache Http Web Server

2.0.47 (http://httpd.apache.org/docs-2.0/) with Tomcat Server 4.1.27-LE-jdk14 (http://jakarta.apache.org/tomcat/) and Cocoon Servlet Technology 2.0.4 installed. Apache Tomcat uses Java jdk 1.4.2. (http://java.sun.com/) All the requests are sent to the Apache Http Server (using the same URL). Then it redirects them to Apache Tomcat Server according to the device used. The processing is implemented using Apache Cocoon Servlet Technology.

Clients used:

- Client 1: iPAQ 3870 pocket PC with Intel Strong ARM SA 1110 (206 MHz) processor, Windows Pocket PC (CE 3.0) operating system and IE 3.02 Pocket Internet Explorer. It connects to the server over WLAN (11 Mbps).
- Client 2: Sony Ericsson P800 phone with the ARM 9 (156 MHz) processor, the Symbian 7.0 operating system and Opera 6.0 / SE R101 Browser for the P800/802 installed. The connection to the server is established through GPRS CS-2 (53.6 kbps).
- Client 3: Deck-It 1.2.3 WAP emulator (http://www.pyweb.com/tools).
- Client 4: Sony Ericsson P800 phone with the ARM 9 (156 MHz) processor, the Symbian 7.0 operating system and his built in browser. The connection to the server is established through GPRS CS-2 (53.6 kbps).

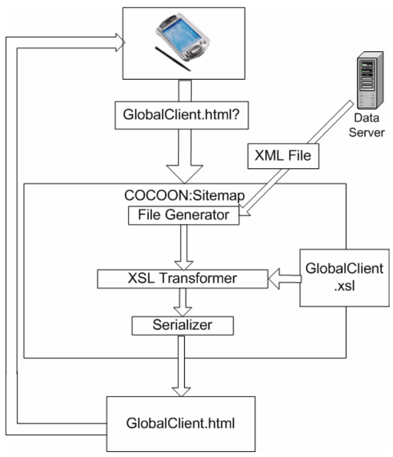


Figure 5. Dynamic content generation

In Figure 5 the process of creating content dynamically in case of the iPAQ pocket PC is shown. The user requests the content for GlobalClient.html. This name is then matched with preset patterns from the sitemap that contains the Cocoon pipeline. The part of the pipeline matching the requested name uses the appropriate XML file and XSL transformation file to generate the content. The content is then serialized and sent back to the client.

V. Case Study: Visualizaton of Grid Monitoring Data

GRID monitoring data is interesting to visualize because of it hierarchical structure and variation of data values. The main data repository for our solution was provided by the MonALISA system [8]. The MonALISA (Monitoring Agents in A Large Integrated Services Architecture) system provides a distributed monitoring service in this case used to monitor AliEn Grid sites (http://alien.cern.ch/). Grid computing has been described as s form of distributed computing that involves coordinating and sharing computing, application, data, storage, or network resources across dynamic and geographically dispersed organizations [5].

Variation of data values indicates the need for dynamic content generation, in our case .html or .wml pages that are adapted for particular devices and browsers. Our goal was to implement a Web based service that would provide a user with a view of the monitored network hierarchy, in addition to the values of various monitoring parameters for different Grid sites. Sites are organized into a hierarchy of "farms" and "clusters", referring to the geographical and/or logical grouping of nodes into virtual computing systems.

Users access the service by entering a unique URL, independent of the device being used. The main page generated dynamically afterwards shows the configuration of farms and clusters in the form of a table. User can choose an interval in which he wishes to see the parameters of clusters and nodes. The other pages are accessed through links from the main page. The links have parameterized URLs that are processed in a similar way as the main page in Cocoon but are matching different names in the pipeline. The parameters passed through the URL are used to request XML files from a service that returns the XML file according to those parameters. The parameter values returned by the service are actual values fetched from a MonALISA database. The three cases of access to the data from three kinds of devices will be shown in the rest of this chapter.

A. iPAQ

The main page for the iPAQ pocket PC (Figure 6) contains the table in which all the farms, clusters and number of nodes in each cluster that are available are shown. On the bottom of the main page there is a form, attached to the script made in JavaScript language, in which the user can choose an interval for which he wants to get the data. Names of farms and clusters are also links to pages with farm configuration or cluster configuration. The parameter values in those pages are given for the same interval as chosen on the main page. If the user wants to view how parameter values change through the period selected he can click on the parameter name to get the graph as shown in Figure 7. The mean values for the according periods are shown on top of the graph.

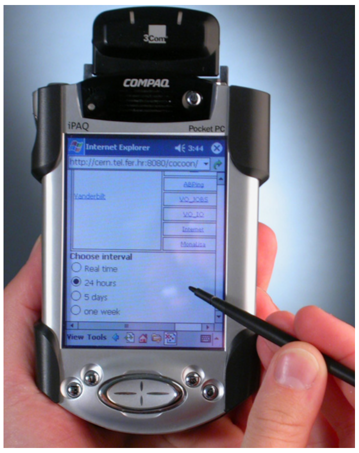


Figure 6. Main page displayed on iPAQ

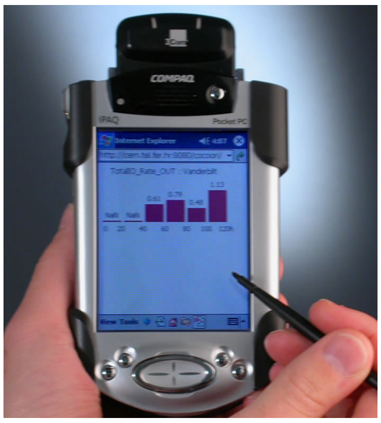


Figure 7. Histogram displayed on iPAQ

B. P800

The content for the P800 phone is accessed the same way as for the iPAQ but the size of the visualization is adjusted to the smaller screen of the P800. As described in section V.A for iPAQ, the main page for the P800 also contains the table in which all the farms, clusters and number of nodes in each cluster that is available are shown. Form attached to the script made in JavaScript language was used to allow the user interval selection. This form is placed on the bottom of the main page. Names of farms and clusters are also linked to pages with farm configuration or cluster configuration. The parameter values that are shown in those pages are given for the same interval as chosen on the main page. If the user wants to view how parameter values change through the period selected he can click on the parameter name to get the graph. The mean values for the according periods are shown on top of the graph (Figure 8).

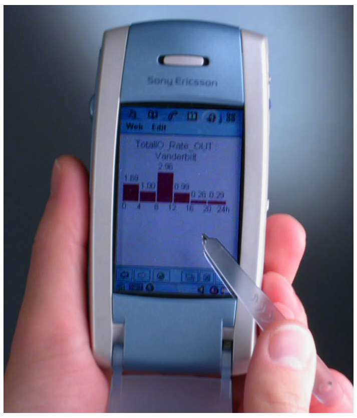
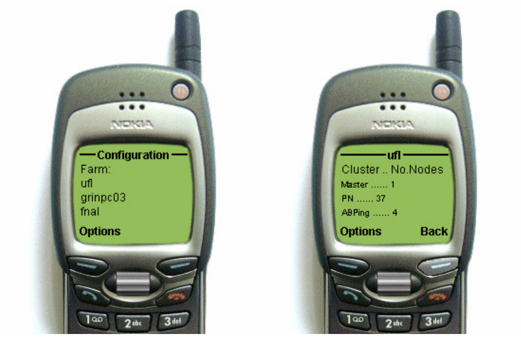


Figure 8. Histogram displayed on P800

C. WAP

In Figure 9 you can see the main page for WAP enabled phones. The user gets the names of all the farms available. Then he can choose the farm he is interested in by selecting the farm name. After farm selection he gets the card containing the list of clusters for that farm and the number of nodes for each cluster as shown in Figure 9. User also can request more details for the chosen farm or he can return to the previous card by pressing the Back button. If more details are chosen the user gets the WML page with parameter names and mean parameter values for all the clusters in the chosen farm. Finally the user gets the list of nodes and available parameter values for the selected cluster.



Configuration
Farm:
ufl
grinpc03
fnal

ufl
Cluster .. No.Nodes
Master 1
PN 37
ABPing 4

Figure 9. a) Main page b) Cluster configuration
Displayed on WAP emulator

VI. Conclusion

We made it possible to access the same content from different devices and to dynamically adapt the visualization of that content to the characteristics of the certain device. Apache Cocoon Servlet Technology enabled us to build our solution in a simple and extensible way. Our implementation supports adapted visualization in the form of graphs and tables as well as transparent access for the iPAQ pocket PC, P800 and WAP enabled mobile phones. The implementation was successfully tested using Grid monitoring data, but it also can be easily adapted for visualization of any kind of hierarchical data structures.

In future work identification of user's terminal and network can be extended with CC/PP or SIP and SDP.

Acknowledgment

This work resulted as a part of FER/ETK Summer Camp 2003 Workshop.

References

[1] N. Medvidovic, M. Mikic-Rakic, N. R. Mehta, and S. Malek, "Software Architectural Support for Handheld Computing", *IEEE Computer 36*, 9, pp. 66–73, 2003.

[2] M. Butler, F. Giannetti, R. Gimson and T. Wiley, "Device Independence and the Web", *IEEE Intenet Computing 6*, 5, pp. 81–86., 2002.

[3] K. H. Britton, R. Case, A. Citron, R. Floyd, Y. Li, C. Seekamp, B. Topol, and K. Tracey, "Transcoding: Extending e-business to new environments", *IBM Systems Journal 40*, 1, pp. 153–178, 2001.

[4] Y. Hwang, J. Kim, and E. Seo, "Structure Aware Web Transcoding for Mobile Devices", *IEEE Internet Computing 7*, 5, pp. 214–221, 2003.

[5] I. Fosrer and C. Kesselman, eds., *The Grid: Blueprint for a Future Computing Infrastructure*, Morgan Kaufmann, 1999.

[6] Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation 6 October 2000 [On-line: http://www.w3.org/TR/2000/REC-xml-20001006]

[7] XSL Transformations (XSLT) Ver. 1.0, W3C Recommendation 16 November 1999 [On-line: http://www.w3.org/TR/1999/REC-xslt-19991116]

[8] H.B. Newman, I.C. Legrand, P.Galvez, R. Voicu, C. Cirstoiu, MonALISA: "A Distributed Monitoring Service Architecture", *Proceedings of 2003 Conference for Computing in High Energy NuclearPhysics*, 8 pp., La Jola, California, 2003.

[9] Composite Capabilities / Preference Profiles: Requirements and Architecture, W3C Working Draft, July 2000 [On-line: http://www.w3.org/TR/CCPP-struct-vocab/]