WAP-G: a case study in mobile entertainment

Conference Paper · February 2002

DOI: 10.1109/HICSS.2002.994015 · Source: IEEE Xplore

2 authors, including:

Thomas Mück
University of Vienna
71 PUBLICATIONS **229** CITATIONS

SEE PROFILE

WAP–G: A case study in mobile entertainment

Markus Hagleitner and Thomas A. Mueck
Institute for Computer Science and Business Informatics
Universität Wien
A-1010, Rathausstr.19/9, Vienna, Austria
{hagleitner | mueck}@ifs.univie.ac.at

Abstract

WAP-G stands for a recent (4.Qrt 2000) real-life case study dealing with the practical aspects of large scale mobile entertainment services as currently prototyped by European internet service providers.

Software platforms for mobile entertainment services, e.g., interactive multi-player games or profile based personal matching using real-time geographical information, are facing a number of software-technical challenges, with respect to both, reliability as well as scaleability.

In this case study report, we will provide a brief outline of the specific requirements for the WAP-G platform and, in the main part of the paper, focus on two technical key issues in the context of interactive multi-player WAP games: maintaining consistent user session and game state information in presence of unreliable WAP connections and device independent output generation and transformation for not fully standardized WML end user devices.

The former issue has been dealt with by running an array of server-side Mealy-style state machines yielding a consistent game state database used for session and game recovery in presence of flapping WAP connections. The latter challenge has been met by using an XML-style intermediate output representation based on a specific WML DTD in such a way that XML output generation and translation happens as a side effect of state transitions in the above mentioned state machines. In this paper, both both problems together with real-life tested solutions are discussed in detail.

1. Introduction

Despite of an ongoing transition to the GPRS mobile transmission standard, today's wireless data networks as operated by European telecom service providers are still based on GSM. Such networks provide a rather constrained communication environment compared to state of the art wired networks: because of their inherent technical limitations, wireless data networks tend to have: less bandwidth, higher latency, less connection stability and less predictable availability.

Although GPRS will improve the situation with respect to bandwidth and network latency significantly, the practical impact on stability and availability is still to be evaluated. For interactive text based services with rather low data transfer volumes, e.g., the interactive games based on the Wireless Application Protocol (WAP) as discussed in this paper, limitations with respect to connection stability and service availability are considered more severe than a lack of bandwidth or guaranteed response time. Additionally, the micro browsers delivered with WAP enabled mobile end user devices (e.g., cell phones) feature a significantly smaller common core functionality than traditional html browsers.

Consequently, we focus in this report on session and game recovery by means of finite automata (see [5] for background information and [8] for a particular application in the database realm) and on device independent output generation.

This case study about different approaches to implement mobile services (WAP-games) was done in cooperation with a private start-up company, which planned to offer mobile entertainment services on the European continent. The goal was to design and implement a first prototype of a particular WAP (Wireless Application Protocol) –service, using the current mobile technology. Beside that, a generic prototype-framework was developed building a basic layer for implementing specific mobile services in a JAVA-based programming environment.

2. WAP-G Project Requirements

The current base-technology for mobile Internet services is the Wireless Application Protocal (WAP 1.2). The system prototype was implemented using a JAVA 2 - programming environment, different JAVA standards and JAVA-APIs like JAVA-Servlets, JSDK-session management, JAVA-RMI and JDBC. For dynamic output

0-7695-1435-9/02 $17.00 (c) 2002 IEEE

generation, XML was chosen to represent the output data in form of general document types, which are finally translated via XSLT[1] into either a specific WML[2]-dialect or HTML.

Several non-functional requirements have been stated by the service provider prior to actual prototype development work. In the following, a brief overview of the most important system requirements is given.

- Session/state -management and -recovery

Since there is a need for user interaction a session-handling mechanism and accordingly a suitable state-management is needed. The wireless application protocol is based on - and uses extensively - existing communication-protocols like HTTP 1.1, which is a stateless protocol. Session recovery in case of possible failures is becoming more important for wireless networks as they are less reliable and stable. The system should take the responsibility to recover a user's last state, if there was a connection-loss or any other problem on the way between the service and the end device, which lead to a loss of an user-session. Similar requirements, i.e., resuming open sesssions upon reconnection after a forced disconnet are described in [14].

- End-device independence

Any successful mobile service provider essentially has to support and provide content which is suitable for every possible browser on the market. Handling two or three different major Web-browsers is already difficult, because of their incompatibility in some details to the W3C[3]-specifications or proprietary standards, see also the material presented on the Web Accessibility Initiative (WAI), http://www.w3.org/WAI . However, the situation is significantly worse in the wireless world, where around 50[4] different WAP-enabled devices exist, which - of course - include different micro-browsers. In [13], several problems concerning platform independence are addressed.

This and the fact that there is much more coming up in the near future, makes it essential to implement a really device-independent output generation, which finally can be adjusted to a particular micro-browser by transforming the output to a special WML-dialect, using XSL-stylesheet technology (see e.g. [15] for considerations on context aware browser and device optimized presentation of content).

- High reliability

It is extremely important to have a highly reliable system. The main problem at this point is the - at the moment - very unstable wireless network. This is one of the requirements directly addressed in this paper.

- User - to User interaction

Not only interaction with the gaming system itself is essential, but also with other users of that service. This requires synchronization of user-actions and sometimes automatic refresh of the displayed service content, without any user-input.

- Generic framework to create special purpose services on top of it

It is important to have a purely generic software framework which provides all the basic functionality, in order to make it easier to create new mobile applications. Basically, it should act as an abstraction layer, which hides complexity of the underlying technology from the developer of mobile services, who has to concentrate on the service content, rather than thinking about technical issues.

- Persistent data storage

Last but not least, the framework must include a mechanism to store game data persistently. Addititonally, for some special services the application developer will have a need for saving data persistently in a database, e.g. for personalization of a service, or even for a whole business process.

3. Runtime Environment and Request Processing

The overall runtime environment is shown in Figure 1. Requests in this evironment are processed following a simple bidirectional pattern. Referenced tags to be found in Figure 1.

- any mobile device or HTML-browser is creating a HTTP-request (tag 1), which is sent over the wireless network infrastructure to a WAP-gateway. This gateway is usually operated by the operator of the mobile network (tag 2).
- the WAP-gateway produces a HTTP-Request and sends that request to a server, which is identified by its URL. In the case study at hand, an Apache Web-Server 1.3.12 was used (tag 3).
- The server-side running Java-application (Java-servlets) processes these requests, calls the business logic, and produces some application-dependent XML-output, which is transformed by using XSLT and a XSLT-processor into another XML-format (e.g. WML) or HTML-format. The used stylesheet (XSLT) is chosen dependently on the USER-AGENT. The

1. XSLT: Extensible Stylesheet Language Translation
2. WML: Wireless Markup Language
3. W3C: World Wide Web Consortium
4. Approximate number of different WAP-enabled mobile handsets in March 2001

Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS-35'02)
0-7695-1435-9/02 $17.00 © 2002 IEEE

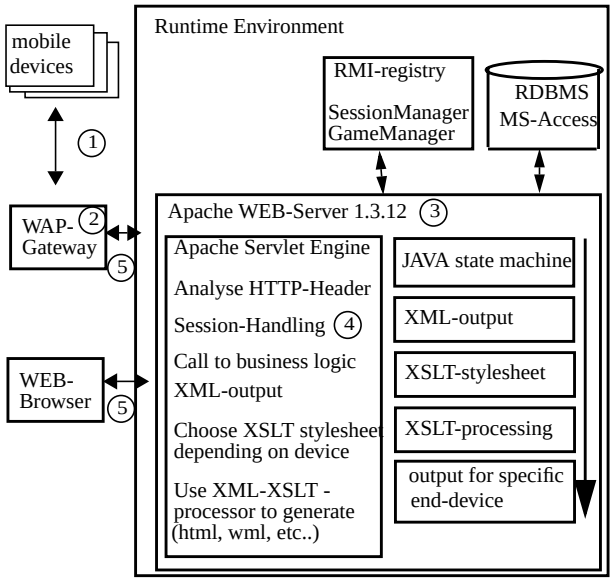

Figure 1: System Runtime Environment and Request Processing Schema

device-type is extracted from the HTTP-header (tag4).

- The servlet is also responsible for session handling and answering using an HTTP-response (tag 5).

Within this servlet-engine almost all of the server-side functionality (the Java-framework and also business logic) is implemented and is actually running inside this Apache module. Additionally, there are also two system-elements, which are running outside the servlet-engine:

- an MS-Access database, which is accessed via JDBC/ ODBC-bridge.
- an RMI-registry, which is the home of some manager-objects. It is running on a Java virtual machine of its own.

4. Managing user sessions and user states

One of the most important requirements for the framework was state-management for multiple users. In particular, the system has to

- receive and handle HTTP-requests from several concurrent online-users
- handle for each user the current state in the system to facilitate redo actions and
- recognize a possible transient connection loss and recover the user's last state upon succesive reconnection of the user to the gaming system.

From a formal point of view, in this context a user is interacting with an abstract reactive system. Viewed as some sort of black box, the system determines legal user actions (i.e., the set of possible user actions) based on an internal data set containing the current user state and the current user input (or, in general, any other input from the environment).

This state-change-mechanism is usually seen and modelled by means of finite state automata. In our case we can use the Mealy machine model (see [5]) to show how state changes triggered by external events in reactive systems can be described and formalised. In the following subsection, this model is briefly described.

4.1. Generalized Mealy Machines

A Mealy machine is a finite automaton extended by an additional output function. This can be used to model reactions (state changes and their side effects – in our context outputs to mobile devices) on a particular input sequence.

The definition of a Mealy Machine ([5], standard text on automata theory) is seperated into two parts, defining

- a *Generalized Mealy Machine Specification* (GMMS) which can be seen as template, which is used to generate Mealy machines using a simple instantiation mechanism
- a Generalized Mealy machine (GMM), which represents one particular instance of the Mealy machine specification, in our case a session, a game, and so on.

Definition 1: GMMS and GMM

Let Q, Σ and Γ denote three pairwise disjoint finite sets, called the state set, input alphabet, and output alphabet, respectively.

A generalized Mealy machine specification (GMMS) defined over Q, Σ and Γ is characterized by a partial function λ. This function used to map the machine´s state set and input alphabet into the state set and into strings over the output alphabet.

$\lambda: Q \times \Sigma \rightarrow \theta \times \Gamma^*$.

A generalized Mealy machine (GMM) is an extension of a particular GMMS which is in a current state, say θ. Consequently, a generalized Mealy machine is fully described by an ordered pair (θ, λ) with $\theta \in Q$. EOD

A GMM in current state θ receiving a symbol $\sigma \in \Sigma$ enters the specified successor state and produces a specified string of output symbols. This reaction can be described in a denotational fashion by an ordered pair $(\theta_{next}, \gamma) = \lambda(\theta, \sigma)$, with $\gamma \in \Gamma^*$ if and only if λ is defined for (θ, σ). If and only if λ is not defined for (θ, σ) the machine does not react at all. EOD

Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS-35'02)
0-7695-1435-9/02 $17.00 © 2002 IEEE

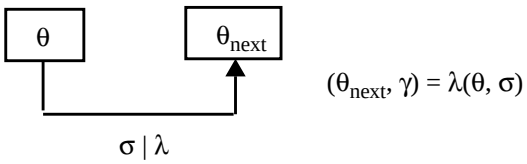

Figure 2: Visualizing a state transition

State transitions diagrams, in an aggregated representation (see [7] and [9]) also called Statecharts visualize the structure of l in a graphical form, see Figure 2. It describes the behavior and the reactions of a system in terms of system states and the corresponding state changing events which are to be sensed by the system.

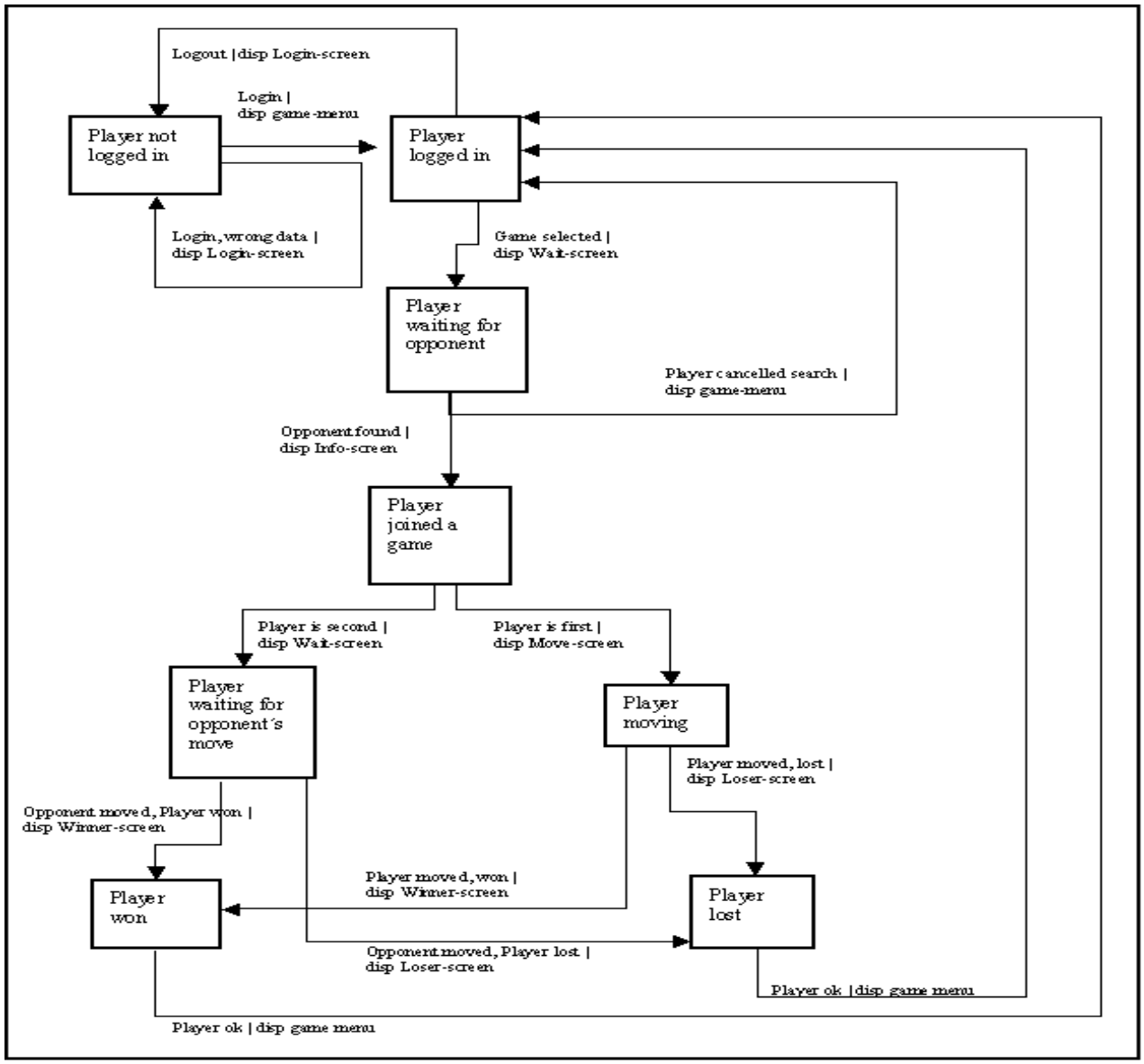

Figure 3: Sample statechart visualizing the main control states of a player

4.2. System reactions formalized by a GMMS - an example

The following sample state-transition diagram in Figure 3 shows the main user-states in the prototype. Since it is a sample diagram, it does not show all states in detail. A specific game application would have to be presented in another statechart, which visualizes the game itself.

The statechart shows all possible states and transitions for a service-user under following simplifying assumptions:

- a user has to login to the system first
- the user already selected a game
- the available games are synchronized two-player games which means that the players move one by one until a win- or loose-situation occurs (Chess, Tic-Tac-Toe, etc.)

Two approaches how this kind of finite automata can be implemented, will be discussed in the following section.

5. System Architecture

In order to achieve the system requirements, five main system actors have been identified. The process leading from an incoming HTTP-request to the HTTP-response is shown in Figures 4 and 5 below. However, state transition control can be implemented by two fundamentally different concepts called Centralized State Control (CSC) and Decentralized State Control (DSC), respectively. Below, the two design alternatives and the decision made for the prototype implementation are briefly described.

5.1Centralized State Control

One session handler, implemented as Java-Servlet, listens for all incoming HTTP-requests on a server. It receives and parses the incoming requests and is able to extract the input symbols ($\sigma \in \Sigma$) out of it (e.g. user id and the user's input). The session manager (generic state machine) performs the state-changes $(\theta_{next}, \gamma) = \lambda(\theta, \sigma)$. It also remembers every user-state θ, which enables the system to recover lost user-states for a particular user on reconnection.

The business logic is the part which has to be implemented by the service-developer and is not part of actual state machine. In fact, the game (business) logic yields the states and transitions for a special service (e.g. game) and therefore configures the generic state-machine.

The output generation is that part of the class library which is responsible for the output function ($\gamma \in \Gamma^*$), which in our case is a predefined defined XML-document type format (see next section). The output manager sends the result of a request back as HTTP-response. Most important is its role as transformer of the output to a user-agent dependent markup language.

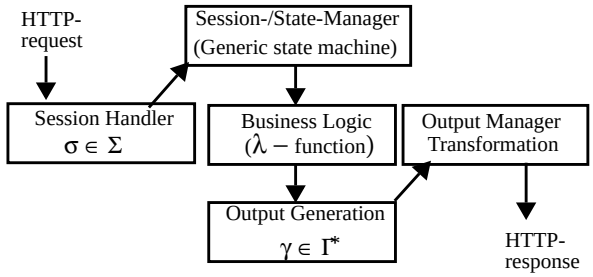

Figure 4: System actors and flow of request-processing – CSC

5.2. Decentralized State Control

The difference to the CSC design layout is that there is not only one SessionHandler, which listens for incoming HTTP-request and forwards them to the state-machine, which implements the λ-function of the Mealy machine, but there is a Java-Servlet for every possible user-state, see Figure 5.

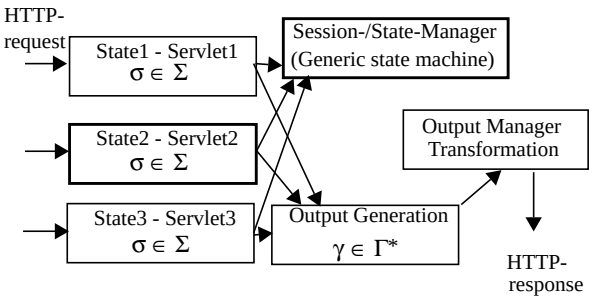

Figure 5: System actors and flow of request-processing – DSC

The HTTP-request is routed to the matching user-state-servlet, which performs the matching transition $\lambda(\theta, \sigma)$ and informs the state-machine (Session/State-Manager) about a state-change. The output-process remains the same and will be discussed in the next section.

5.3. Design Decision CSC versus DSC

The main reason to use the DSC layout (i.e., the alternative described in Subsection 5.2) would be the bottleneck on the entrance to the system in the CSC layout, which initially yielded severe performance

Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS-35'02)
0-7695-1435-9/02 $17.00 © 2002 IEEE

problems when serving many concurrent users at a time during early test runs.

On the other hand CSC fulfills very well the requirement to have some kind of abstraction layer for mobile service developers, who do not want to think about the underlying state transition mechanism. In this context, CSC provides a high level abstraction from the underlying processes and is therefore much easier to handle for service developers, as they only have to specify and implement the λ-function. DSC in turn appears less maintainable and provides more room for errors, as every state is implemented as one Java-Servlet.

Considering these arguments, CSC was finally chosen to be implemented in the prototype, as the performance

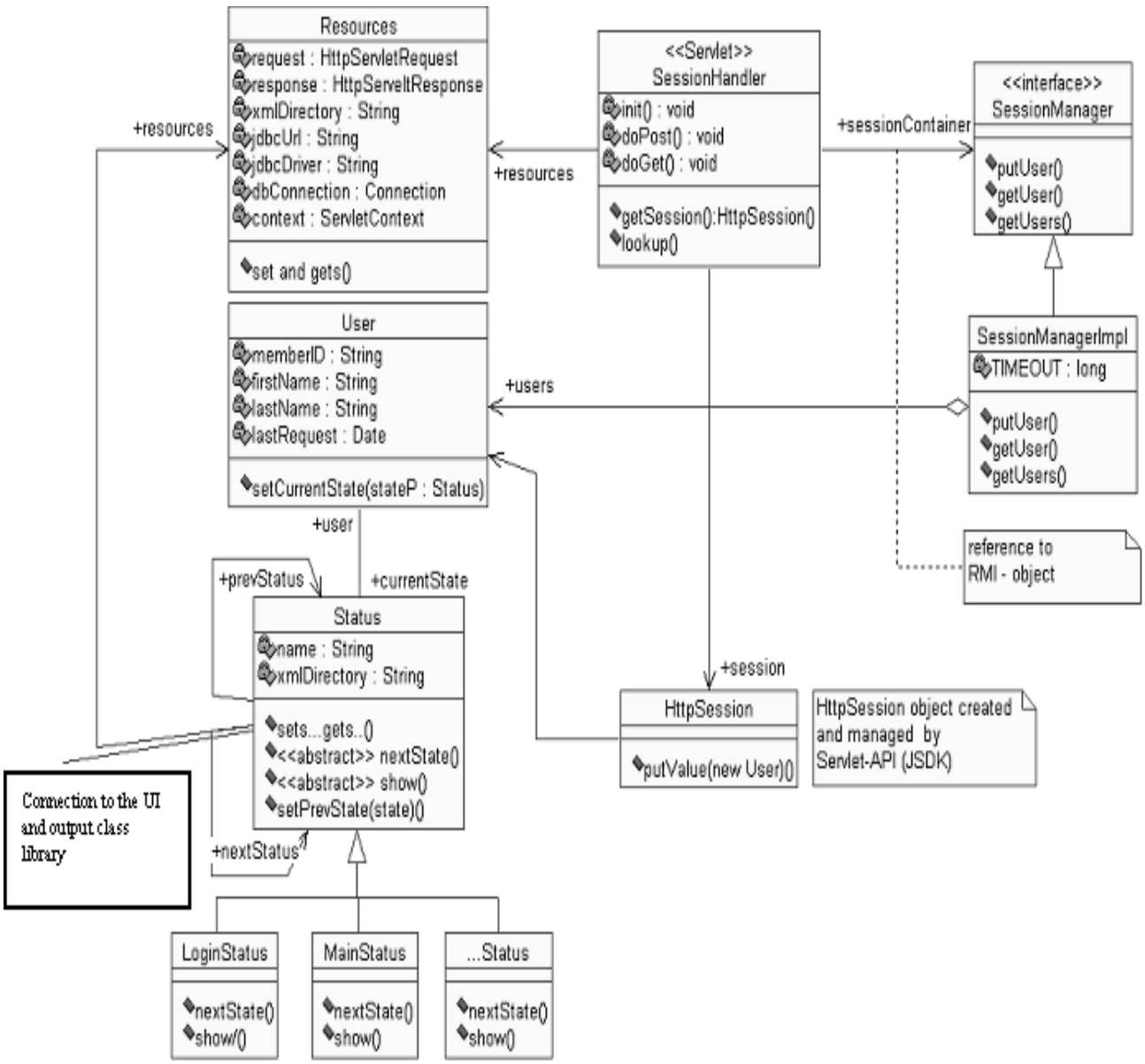

Figure 6: Class diagram for session/state management and -recovery

problem was assumed to be solved by software- and hardware-tuning in a later stage.

5.4.Class library for session and user state handling

The class-diagram in Figure 6 shows an excerpt of the complete class library and will help to understand the structure and collaboration between the objects.

A class called *SessionHandler* extends *HTTPServlet.* This servlet is the single entry point from the mobile network into the system. This means, that every URL, referring to this prototype, is containing (directly or indirectly) a reference to this class. The *SessionHandler* can extract information like user-agent, etc. and different parameters from the HTTP - header via the *HTTPRequest* - class, also provided by the JSDK API. Most importantly it is able to find out a session identifier. A user object represents the end-user. This object contains information like memberID, password, name and so on. Furthermore it contains a reference to the user's current *Status*-object, called *currentState*.

Status is the base class for every possible state, a user can have in the whole system. If the user comes into the system, he or she has to login first. Therefore the user is in the *LoginStatus*. One can find this class as a derived class from *Status*. In the class *Status* there are two methods, which are declared abstract. A specific state is displaying its contents to the user with *show()*, *nextState()* decides which state is the next one and performs also all side-effects. When the user logs in, a user-object is created, containing a reference to his/her first state. After that the user-object is put into the session-object. For each request, the JSDK API can find out the right session-object, belonging to a particular session (or user). This happens via URL-rewriting or cookies.

The session-object can provide the reference to the user-object, so that the state-machine can continue to do its work.

Another task of the *SessionHandler* is to fill an instance of the class *Resources*, which provides all the other classes with resources of every type. From database-connectivity to global servlet-parameters, one can get any global resource or information.

Last but not least there is the RMI-interface to the *SessionManager*. A server-object, which holds a vector of all currently logged-in users. The *SessionHandler* uses the remote methods to update the list of currently logged-in users, to have in case of a connection failure the possibility to recover the last state of a user. The framework manages a list of currently logged-in user-objects. User-objects also contain the user's last status in the system. Therefore the list always represents the latest system-state, which enables the framework to recover lost sessions as described above. It would also be possible to store this data in a database. The solution using an RMI-server object was chosen, because of its simple interface and the possibility to store whole Status-objects rather than flat data.

6. Output - generation and -transformation

Another requirement was *device-independent output generation*, a very important one, because of the rapidly developing and innovative mobile handset-market. Every month we can expect two or three new WAP-enabled handsets to be announced on the mobile market. Therefore one of the challenges of this project was to find the most suitable technology, providing the required independence.

The second goal of this part of the project was to design a simple class-library, to create WML-user interfaces in a comfortable way. WAP-application-developers, who are familiar with the WML-basics, should be able to use it without knowing much about the underlying technology.

6.1. Using XML as intermediate representation

XML (eXtended Markup Language) was chosen as most flexible way to create structured information, which can be transformed into any handset specific markup-language on the fly. The current approach to maintain flexibility in creating output for mobile communication is the extensive use of XML and XSLT.

Basically, XSLT is a standard way to describe how to transform (change) the structure of an XML - document into an XML document with a different structure using stylesheets. It can be thought of as an extension of the Extensible Stylesheet Language (XSL).

Consequently, also in the WAP-G project XSLT is used to describe how to transform the *source tree* or data structure of an XML document into the *result tree* for a new XML document, which can be completely different in structure. The coding for the XSLT is also referred to as a style sheet and can be combined with an XSL style sheet or is used independently.

Since WML can be regarded as an XML-dialect, we are able to use this technology to generate different WML-styles (Stylesheets) for each mobile device, which has to be supported.

The prototype uses the implementation of the XML-parser and the XSLT-processor from the XML-project of the Apache Group. Apache's XALAN - stylesheet processor is available in the version 1.1 and includes also Apache's XERCES - XML-parser. The state-machine generates output-symbols ($\gamma \in \Gamma^*$) in form of a standard XML-file-format, specified by a Document Type Description (DTD). This file is parsed and validated by the

Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS-35'02)
0-7695-1435-9/02 $17.00 © 2002 IEEE

XML-parser. A stylesheet has to be written for every different browser (e.g. WEB-browser, WAP-browser), which is the information source for the XSLT-processor, to know, how to restructure the XML-file into another language like HTML or WML.

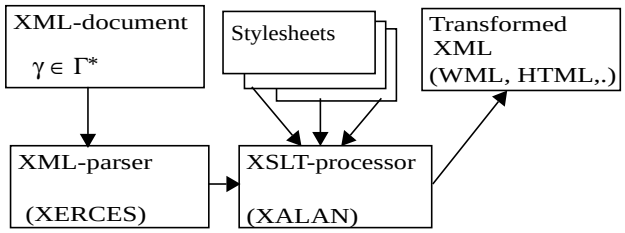

Figure 7: Standard XSL-Transformation process

6.2.. Functionaliy and features of the output generation module

As usual in a Mealy machine, output is generated as side effect of state changes in the state machine. The state-machine provides a standard XML-output-file, defined by a DTD.

The first part of the class-library, depicted in Figure 8, provides the WAP-developer with an easy interface to generate the output by hiding the whole tag-based output process:

The base class for the XML-Output is called *XMLPage*. *XMLPage* includes the basics for every possible screen (e.g. the XML-directory, where the output files are created and so on...). It implements a *show* - function and defines an abstract member function called *createXMLData*(). *XMLPageGeneralScreen* defines the

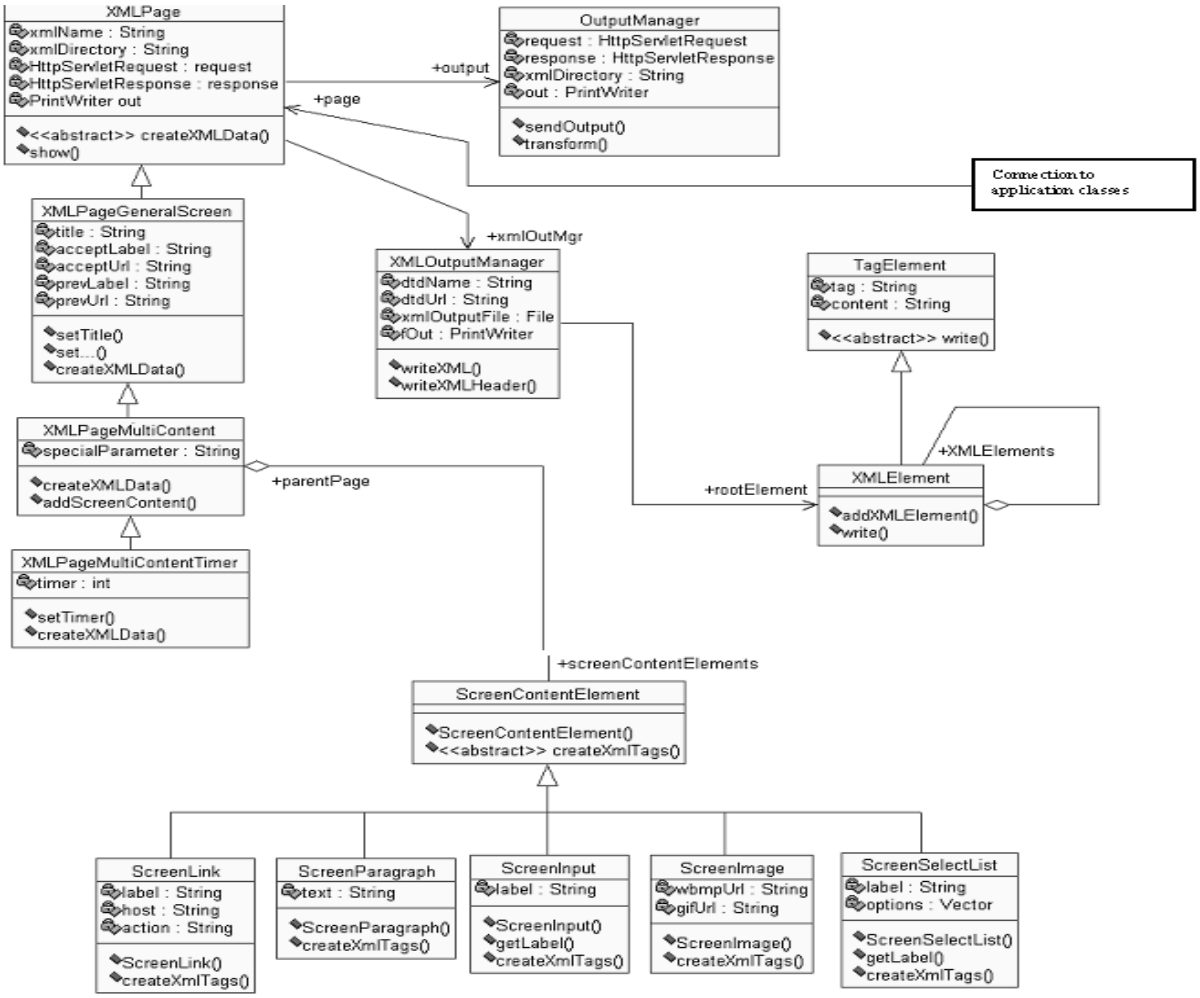

Figure 8: Class diagram for output generation module

Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS-35'02)
0-7695-1435-9/02 $17.00 © 2002 IEEE

basic structure of a XML-file for WML-pages. There is always a *title* and two *buttons*, called *accept*, and *prev*, which refer to the WML-specification. *XMLPageMultiContent* and *XMLPageMultiContentTimer* are finally those classes, which define the two most used types of screens in the whole project.

To be able to contain multiple content, *XMLPageMultiContent* implements a vector of *ScreenContentElement*, which is an abstract class defining the basic interface of screen elements. The function *createXMLTags*() has to be implemented by each subclass. For every major WML-element there is a *Screen<WML-element>* class. The function *createXMLData*() of *XMLPage* creates an object tree, which represents the XML-output tree. Writing this XML-tree in a physical file on a disk is the task of *XMLOutputManager*.

XMLPageMultiContent and the *ScreenContentElements* use *XMLElement* to define their tag-based structure. *XMLElement* is a subclass of *TagElement*. *TagElement* defines that there is always a start tag and an end tag. In between there can be any other element (content or data). We need this knowledge for creating a XML-file, which is tag-based. *XMLElement* has to implement the method *write*(), which creates this <tag>content</tag> structure, where content can include an indefinite number of other elements.*TagElement* and *XMLElement* were designed using the composite - design pattern[1]. A subclass of an abstract superclass can contain an indefinite number of the same or other subclasses of the superclass to represent a composition of any type (in this case, the composition is a XML-tree of XML-elements), because all of them implement the same interface.

In this case, the interface only consists of one function called *write*(). This enables the framework, to have just one entry point, namely *rootElement*, where we call *write*(), and the whole tree is generated in one single step.

XMLOutputManager opens an output stream to a physical file on the disk and calls *write*() for the rootElement (which is the root for a whole WML-screen). Before that, it also generates the header of the XML-file, which includes the definition of the used DTD for this particular XML-document type.

6.3 A document type definition for WML user interfaces

The DTD in Figure 9 defines a valid structure of a XML-document. In the example below, the root-element of a WML-screen is named *screen*. It defines, that a screen consists of a *title*-, a *submitURL*-, a *content*-, an *acceptButton*- and a *prevButton*-element. These elements can consist of other elements. For example *content* is everything, the user can see, between the screen-title and the buttons.

1. Composite-pattern see [6] p. 163

```
<!ELEMENT screen (title, submitURL, content, acceptButton, prevButton)>
        <!ELEMENT title (#PCDATA)>
        <!ELEMENT submitUrl (#PCDATA)>
        <!ELEMENT content (paragraph,textline,input,link,image,selectList,multiSelectList)>
                <!ELEMENT text (#PCDATA)>
                        <!ELEMENT paragraph (text)>
                        <!ELEMENT textline (text)>
                        <!ELEMENT input (label, defValue, format)>
                                <!ELEMENT format (#PCDATA)>
                        <!ELEMENT link (label, url)>
                        <!ELEMENT image (label, wbmpUrl, gifUrl)>
                                <!ELEMENT wbmpUrl (#PCDATA)>
                                <!ELEMENT gifUrl (#PCDATA)>
                        <!ELEMENT selectList (label, options, defValue)>
                        <!ELEMENT multiSelectList (label, options, defValue)>
                                <!ELEMENT options (option)>
                                        <!ELEMENT option (#PCDATA)>
                                <!ELEMENT defValue (#PCDATA)>
                                <!ELEMENT acceptButton (label, url)>
        <!ELEMENT prevButton (label, url)>
                <!ELEMENT label (#PCDATA)>
                <!ELEMENT url (#PCDATA)>
```

Figure 9: DTD for WML interfaces

Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS-35'02)
0-7695-1435-9/02 $17.00 © 2002 IEEE

7. Conclusions

This case study shows that the rather simple Mealy-Machine model bears sufficient expressiveness to model and describe rather complex reactive systems. Furthermore, the states itself, including state changes and output symbol generation was simple to implement, since we created first a framework, which implemented the generic state-machine.

This kind of state management also provided us with the opportunity to recover a user´s last state, even when they lost their session, which happens still quite often in current wireless networks.

XML (and friends) is at the moment the best possibility to handle the problems occuring with the many combinations of possible mobile devices, their browser versions and the WAP-gateways they are using. Almost every combination has its effects on the user-interface style. The weak commitment to WAP-standards of some vendors makes difficult for developers of mobile services. Nevertheless XML is powerful enough to handle such difficulties and also showed its power in this prototype implementation.

A further interesting field of research would be the implementation of an editing-tool to maintain and manage XSL-files in a user-friendly way.

8. References

[1] Apache Jserv – servlet engine
http://java.apache.org/jserv/

[2] XML – project group of Apache (XALAN, XERCES)
http://xml.apache.org/

[3] Asbury St., Weiner, S.R., Developing Java enterprise applications, Wiley computer publishing, 1999

[4] Behme H., WML: XML-Dat(ei)en aufs Handy bringen, iX 4/2000 (in German)

[5] Eilenberg, S. Automata, Languages and Machines (Vol. A). Academic Press, Boston, 1974

[6] Gamma E., Helm R., Johnson R., Vlissides J., Design-Patterns, Elements of Reusable Object-Oriented Software; Addison Wesley,1995

[7] Harel D., On Visual Formalisms, CACM 31(5), 1988

[8] Mueck T., Design Support for Initiatives and Policies in Conceptual Models, HICSS94; IEEE CS Press, 1994

[9] Mueck T., Active Databases - Concepts and Design Support in: Advances in Computers, Vol. 39, Academic Press, 1994

[10] Developer's Guide, NOKIA WAP Toolkit, 1999, http://www.forum.nokia.com

[11] WAP Architecture, http://www.wapforum.org, SPEC-WapArch-19980430.pdf

[12] Professional Java Server Programming, Wrox Press Ltd, 2000

[13] G. Austaller, A. Hartl, G. Kappel, E. Kapsammer, C. Lechleitner, M. Mühlhäuser, S. Reich, R. Rudisch, Gulliver Beans: A Tool for Fine Tuning Data Delivery in E-Commerce Applications, Proceedings EMISA 2000 - Informationssysteme für E-Commerce, pp. 29-40, 2000

[14] A. Hartl, G. Austaller , G. Kappel, C. Lechleitner, M. Mühlhäuser, S. Reich, R. Rudisch, Gulliver - A Development Environment for WAP Based Applications, 9th International World Wide Web Conference, Posters' Track, Amsterdam, May 15-19, 2000

[15] Paxcoe J., Ryan N. and Morse D.: Issues in Developing context-arware computing, in: First conference on handheld and ubiquitous computing (Sep. 1999), H.-W. Gellersen Ed., vol. 1707 of LNCS, University of Karlsruhe, Germany, Springer, pp. 208-221

Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS-35'02)
0-7695-1435-9/02 $17.00 © 2002 IEEE