An Integrated Platform for Mobile, Context-Aware, and Adaptive Enterprise Applications

Gerald Hübsch, Axel Spriestersbach, Thomas Springer, Thomas Ziegert

{huebsch, springet}@rn.inf.tu-dresden.de
{axel.spriestersbach, thomas.ziegert}@sap.com

Abstract: In this paper we present an approach for the device-independent authoring of mobile, context-aware and adaptive web applications. We present concepts for the semantic and syntactic adaptation of web dialogs to heterogeneous devices based on the Dialog Description Language. We describe advanced concepts for the integration of dynamic content and user interaction handling in device-independent application engineering. Furthermore, the software architecture of the Adaptation Framework, a runtime environment for dialog adaptation, is introduced. Finally, we present an adaptive sample application and give an outlook to future research topics.

Keywords: Adaptation, Context-Awareness, Dynamic Content, Mobile Devices, Multimodality, MVC, Fragmentation, Single Source Authoring, XML

1 Introduction

According to a study conducted by IDC Research, the number of nomadic workers spending at least 20 percent of their working hours away from home, their main place of work, or both in Europe will increase to over 28.8 million by 2005. There is a large target group of subscribers for Mobile Intranet/Extranet access [IDC00]. This development is set in motion through the rapidly increasing availability of multiple wireless infrastructures such as public WLAN hotspots, GPRS, HSCSD, CONNEXION [Co04] and the startup of UMTS services in Europe. Another major driver for this development is the omnipresence of connected mobile devices like notebook computers, PDAs, and data-capable mobile phones in today's companies. However, the heterogeneity of these devices can be seen as one of the main obstacles for their pervasive deployment in enterprise scenarios. Mobile devices differ in a wide spectrum of parameters. These include various input methods such as keyboard, stylus, numeric keypad and combinations thereof, display size and resolution, supported bearer types, processor speed, power consumption, attached peripherals, browser type, and supported media formats. This set of pa-

rameters together with application- and user-specific parameters forms the context of use for a mobile application. Web-applications development is still mainly focused on desktop-style target platforms with fully-fledged browsers and neglects the additional requirements and limitations imposed by the context of use. Automated adaptation to the context of use represents a promising solution to overcome the increasing differences in mobile device technology while reducing the overall costs and development efforts for multi-device support. Vital requirements for web application authoring languages are concepts for dynamic content generation and user interaction handling. Dynamic content support provides means to include backend data into the content delivered to the client. User interaction handling provides means to monitor and evaluate user interaction events by triggering event-specific calls to application logic. Both concepts require a dedicated runtime platform support. In the following we present an integrated adaptation platform for device independent authoring and provisioning of context-aware mobile applications based on the single-source authoring approach. Furthermore, novel language and platform concepts for dynamic content generation and user interaction handling in device-independent application engineering are presented.

2 Related Work on Adaptive Web Applications

Various approaches have been proposed to address the challenges of adapting web applications to heterogeneous devices. The process of re-engineering a web-application for every target platform is referred to as Manual Adaptation. While this approach yields high quality results, it induces high development and maintenance costs due to the large number of different versions of a single applications. Supporting a new device always obliges a new version of the application. Other approaches like M-Links [ScTr01] and Digestor [BiSc97] suggest the use of transcoding proxies. Transcoding proxies are placed between the mobile device and the content source. They apply heuristics to extract and adapt existing content to the requirements of a specific device. The utilization of heuristics and the lack of meta-information about the adapted source make these approaches too error-prone for the adaptation of complex web applications. An approach for the external definition of machine-readable meta-data for adaptation control using RDF is described in [HoKo00]. Although it enables the expression of fine-grained meta-information such as the importance and role of arbitrary elements in HTML source documents, the external definition of meta-information bears consistency issues if the described source is dynamically generated or changed without notice. UIML [AbHe00] allows the description of user interfaces on an abstract level separating content, structure and style. UIML thus basically enables device independent content description, but allows only for purely syntactic mappings that we have found to be insufficient. UIML does not support meta-data annotations.

3 Concepts for Dialog Adaptation

In the following, the concepts of semantic and syntactic dialog adaptation based on the Single Source Authoring approach are introduced. Semantic adaptation precedes syntactic adaptation. Alongside, our approach for a device-independent dialog description language realizing these concepts is presented by introducing the Dialog Description Language (DDL).

3.1 Single Source Authoring

Single Source Authoring uses a single, generic content description that abstracts from the final content presentation in a specific target markup. Dedicated adaptation algorithms are applied to generate device-specific content representations (cf. 1). These algorithms are parameterized by context information. Additionally, semantic meta-information about the processed content is evaluated by these algorithms. The adaptation algorithms can be deployed directly on the content server or a proxy server.

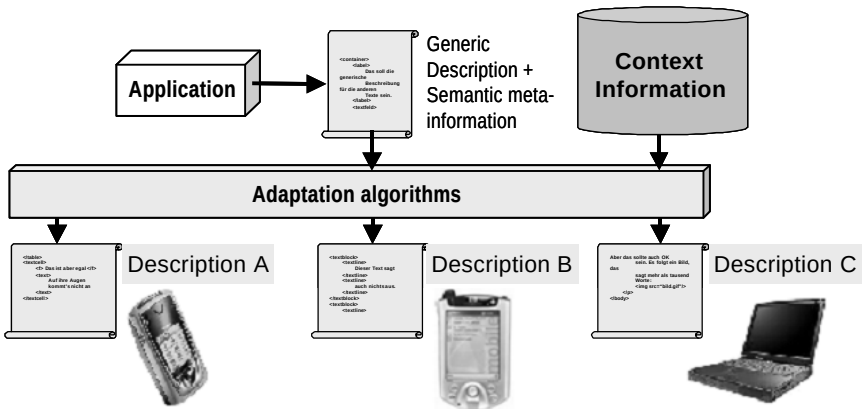

Figure 1 – Single Source Authoring

3.2 The Dialog Description Language (DDL)

We have developed the Dialog Description Language (DDL) for our Single Source Authoring approach. The DDL is a single-source authoring language that enables the separation of structure, presentation, and content of web application dialogs and supports meta-data concepts for semantic adaptation. The DDL is based on a simple XML meta-language model consisting of four basic elements:

<ddl>, *<dialog>*, *<part>* and *<property>*. The <ddl> element is the document root element of any DDL dialog, <dialog> encloses the content section of a DDL document. The <part> element reprensents an arbitrary dialog element. It is utilized to generically describe element types (e.g. input fields, frames, tables etc.), i.e. the element semantics, using a set of properties. It can be nested to allow for ancestor-child relationship modelling. Properties are assigned to parts using the <property> element. A *type property* defines the type of each part (cf. 2). All other properties are type-specific, e.g. for describing the URL of a hyperlink element or the caption of a text input field. Part properties are distinguished by a name attribute.

We have defined semantics, i.e. property sets, for the part types listed in table 1. Note that there is no target markup mapping defined yet.

Part types	**Semantics**
frameset, frame	frameset and frame description
container	description of element groups
table, head, row, data	table elements for describing tables, table headers, table rows and table cells
label	text and text style description
image	inline image description (image source URL etc.)
form	form description (action URL etc.)
submit	form submit element description
radiogroup, radiobutton	radiogroup menu description
textinput	form input field for text
select, option	select menu description
checkbox	checkbox item description

Table 1 – DDL part type semantics

Additionally, DDL supports the concept of inheritance. DDL inheritance allows for the definition of part configurations, i.e. for property element value presets. Syntactically, this concept is implemented by the DDL *<class>* element. It is uniquely identified by its *name* attribute. Its children are <property> elements containing preset property values. Parts inherit these properties through a *class* attribute that references the class by name. The inheriting part is assigned all property definitions within the respective class. DDL class definitions can be externalized in library documents and thus be imported and reused in multiple dialogs.

```
<ddl>
 <dialog>
  <property name="title">Stock quote</property>
  <part>
   <property name="type">form</property>
   <property name="method">GET</property>
   <property name="action">getQuote.ddl</property>
   <property name="layout">vertical</property>
   <part class="stockLabel" name="symbol">
    <property name="content">Symbol:</property>
   </part>
   <part class="stockSymInp" name="stockSymbol"/>
   <part class="quoteButton"/>
  </part>
 </dialog>
 <class name="stockLabel">
  <property name="type">label</property>
 </class>
 <class name="stockSymInp">
  <property name="type">textinput</property>
 </class>
 <class name="quoteButton">
  <property name="type">submit</property>
  <property name="default">Get Quote!</property>
 </class>
</ddl>
```

Figure 2 – Sample DDL dialog using DDL inheritance

Figure 2 depicts a simple DDL dialog containing a form to retrieve stock quotes by entering the stock symbol. The form part uses inline definitions for all its properties. In contrast, the <class> elements 'stockLabel', 'stockSymInp' and 'quoteButton' exemplify the inheritance concept. Type assignments to parts are highlighted in bold face.

3.3 Semantic Adaptation

Semantic adaptation utilizes semantic meta-information about the adapted content. Information about dialog semantics is manually added at authoring time. Thus, the programmer has full control over the semantic adaptation. In DDL, it controls dialog adaptation with respect to the concepts of selective content and dialog fragmentation. Selective content eliminates or selects alternative content, e.g. to avoid the delivery of long explanatory texts to a small screen device. Dialog fragmentation is the process of splitting up complex dialog structures into less complex substructures (fragments) and their sequential delivery to the client while maintaining the logical structure of the dialog. Dialog fragmentation is required to meet display size, protocol and usability constraints of mobile devices. As opposed to the delivery of large, complex pages, fragmentation reduces the need for extensive scrolling and reduces transfer and rendering times for the single fragments. Addi-

```
<part test="/client/display='graphical'">
 <property name="type">image</property>
 <property name="src">logo.jpg</property>
</part>
<part test="not(/client/display='graphical')">
 <property name="type">label</property>
 <property name="content">ACME Corp.</property>
</part>
```

Figure 3 – Selective Content in DDL

tionally, we utilize fragmentation to generate specific target-markup constructs for presentation control, e.g. for content distribution over several WML Cards in a single WML Deck. From a technical point of view, fragmentation is indispensable to meet packet size constraints imposed by the WAP 1.x protocol [WAP01].

In DDL, the concept of selective content is implemented by context queries over context profiles. Context queries are formulated in XPath [W3C99a] that can be attached to part and property elements through a *test* attribute. If the query condi-

tion is fulfilled, the respective element is left in the dialog, otherwise it is removed. Figure 3 shows a sample DDL dialog snippet with context queries. The logo image is shown on graphics-capable clients, a short text is displayed on other clients.

To control dialog fragmentation, DDL provides means to define logical dialog units (atoms) that must not be split up by fragmentation. Arbitrary DDL parts can be marked as atoms by adding the DDL '*atom*' property. Some part types such as radiogroups or parts without descendants are atoms by default. Logical dialog units are utilized to group dialog elements that have a semantic relationship. An adaptation algorithm is responsible for the dialog fragmentation at runtime. In our approach, the algorithm determines the weight (i.e. size in transfer encoding, display space required on the client device) for every atom in the dialog. These atoms are assembled to fragments of maximum size w.r.t. the client constraints and interlinked to allow for user navigation. The runtime environment (see sect. 5) is responsible for delivering the requested fragments and for the collection of user input from forms spanning several fragments.

Figure 4 depicts a DDL code snippet that groups a text input field with its caption using an atomic DDL container to prevent their separation by dialog fragmentation.

```
<part>
 <property name="type">container</property>
 <property name="atom">true</property>
 <part>
  <property name="type">label</property>
  <property name="content">Symbol:</property>
 </part>
 <part name=="stockSymbol">
  <property name="type">textinput</property>
 </part>
</part>
```

Figure 4 – Logical dialog unit in DDL

3.4 Syntactic Adaptation

After semantic adaptation, the DDL dialog must be transformed into a target markup language supported by the target platform. This process is called syntactic adaptation. We use XSLT [W3C99b] for this task. Stylesheets have been implemented for transformations from DDL into XHTML, HTML and WML 1.x. Our approach is extensible towards the support for new target markup languages by simply adding a new stylesheet.

4 Dynamic content and application logic interaction

To facilitate the implementation of adaptive web applications in DDL using the MVC pattern, we have identified four major infrastructure components: (1) *resources* implementing server side application logic and backend access, (2) *dynamic content elements* for retrieval and rendering of dynamic data, (3) *eventing* to trigger application logic through user interaction, (4) a *server-side data model* to maintain application-specific state information. In the following, the implementation of these concepts in DDL is described.

4.1 DDL Resources

DDL Resources implement application logic and backend access. DDL Resources are implemented by Java classes. To model the class properties and method signatures used for event handling and backend access, we have introduced the elements <*resource*>, <*method*>, <*param*> and <*field*> in DDL.

The <resource> element represents a Java class. Its *name* attribute contains a unique name to reference this resource within the dialog containing the resource's definition. Its *class* attribute specifies the fully qualified name of the implementing class (cf. 5).

The <method> element is a child element of resource. Every method element describes the signature of a method implemented by its ancestor. Additionally, method results can be written to the data model (see sect. 4.4) using an optional *reference* attribute. Fig. 5 shows two method definitions for *validateLogin* and *getStockQuote*. The *name* parameter corresponds to the name of the implementing method.

The <param> element is a child element of method. Every param element models a single method parameter. Parameter values may either be read from HTTP request parameters or from the data model. A parameter value is identified by matching its *name* attribute against all request parameter names. If no matching

request parameter is found, the parameter value is read from the data model using the element's *reference* attribute. In case of a successful match, the data model is updated with the request parameter value. For example, the validateLogin method defined in fig. 5 expects a mandatory 'passwd' request parameter to be present upon its invocation. If the 'user' parameter is not present, it is read from the data model.

```
<resource name="AppLog" class="app.StockQuote">
 <method name="validateLogin"
    reference="/model[@id='stock']/authenticated">
  <param name="user"
    reference="/model[@id='stock']/name"/>
  <param name="passwd"/>
 </method>
 <method name="getStockQuote">
  <param name="symbol"
    reference="/model[@id='stock']/symbol"/>
  <field name="__stockName"/>
  <field name="__price"/>
 </method>
</resource>
```

Figure 5 – DDL Resource definition

The <field> element (cf. 5) is a child element of method. Every field element models a single data item returned by a method utilized for backend access, i.e. it represents a cell within a database table. In dynamic data sections (see sect. 4.2), placeholders matching the field's *name* attribute value are replaced with the corresponding data item value.

4.2 Dynamic DDL Content

To support dynamic content, we have extended the DDL with the <*dynamic*> element (cf. 6). The dynamic element has two attributes. The *resource* attribute references the DDL resource that implements the backend access method identified by the *method* attribute.

```
<part>
 <property name="type">table</property>
 <part>
  <property name="type">header</property>
  <part>
   <property name="type">data</property>
   <property name="content">Symbol</property>
  </part>
  <part>
   <property name="type">data</property>
   <property name="content">Price</property>
  </part>
 </part>
 <dynamic resource="AppLog" method="getStockQuote">
  <part>
   <property name="type">row</property>
   <part>
    <property name="type">data</property>
    <property name="content">__stockName</property>
   </part>
   <part>
    <property name="type">data</property>
    <property name="content">__price</property>
   </part>
  </part>
 </dynamic>
</part>
```

Figure 6 – DDL Dynamic element

We refer to DDL dialogs containing dynamic elements as DDL Templates. The access method is invoked during dialog processing (cf. 10). For every data set returned by the access method, the content enclosed by the dynamic element is repeated and the contained field references are replaced with backend data. Finally, the dynamic element is eliminated from the dialog.

4.3 DDL Eventing

Means to monitor and evaluate user interaction on the client are indispensable for web application development. A common strategy for reacting to user interaction is to specify UI event listeners and event handlers. Missing or reduced client-side support for UI event generation and event handling on mobile platforms must also be addressed. Therefore, we have supplemented the DDL with concepts for device independent event description and event handling. Our concept is independent from the structure and the order of UI elements in the dialog.

We have introduced the *<listener>* element (cf. 7) to associate interaction events with DDL form controls. Its *event* attribute describes the event type that determines its processing on the server. The *handler* attribute contains a reference to the DDL Resource and the handler method's name. The *priority* attribute controles the processing order if several events are received simultaneously by the server. For event propagation to the server, the listener element containing the event's type, priority and handler is marshaled into HTTP request parameters during the syntactic adaptation of the dialog. Whenever the user issues a request (e.g. selects 'Login' in fig. 7) the related event is triggered on the application server by invoking the associated DDL Resource method.

```
<part name="loginButton">
 <property name="type">submit</property>
 <property name="default">Login</property>
 <listener event="submission"
    handler="AppLog.validateLogin" priority="1"/>
</part>
```

Figure 7 – DDL listener element

4.4 DDL Data Model

The DDL data model provides means to maintain the application state between consecutive client requests. A DDL data model is represented as a well-formed

XML structure enclosed by a *model* element with a unique *id* attribute (cf. 8). It is defined by the application programmer according to application's requirements. Although DDL Resource instances themselves are stateful, the data model provides means to share state information among instances of different DDL Resources through common model references.

```
<model id="stock">
 <authenticated>true</authenticated>
 <name>John Doe</name>
 <symbol>ACME</symbol>
</model>
```

Figure 8 – Sample DDL Data Model

The data model can also be utilized for dialog content control based on the model state. Therefore, we have introduced a *reference* attribute for DDL parts (cf. 9). This attribute holds an XPath expression over the data model that is evaluated to a Boolean value. Parts with unfulfilled conditions are removed from the dialog.

Our concept is similar to the XForms data model approach [W3C03], but is extended towards a closer integration with application logic to support the above-named objectives.

```
<part reference="/model[@id='stock']/
    authenticated[./text()='true']">
 <property name="type">label</property>
 <property name="content">Logged In</property>
</part>
<part reference="/model[@id='stock']/
    authenticated[./text()='false']">
 <property name="type">label</property>
 <property name="content">Not Logged In</property>
</part>
```

Figure 9 – Model state based dialog content control

4.5 Runtime Integration

Fig. 10 depicts the three major steps of request processing within the Adaptation Framework: Event Processing, Resource Processing and Dialog Adaptation.

At first, the Client Identification step adds context information to the client request.

Based on the event-driven MVC design, request handling starts with analyzing the request for marshaled events and the invocation of event handlers (see sect. 4.3). The Event Processor is the controller component of our architecture. It selects the requested event-handler resource instance and utilizes the DDL Resource description (see sect. 4.1) to interact with resources. Event handling resources select the next view by choosing the appropriate DDL Template.

The Template Loader retrieves Dialog Templates from a repository.

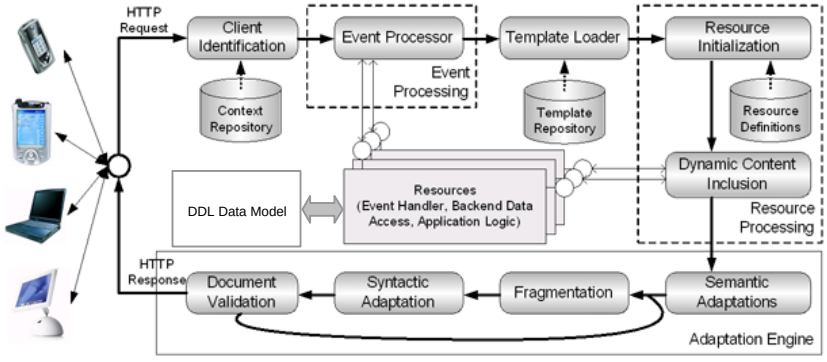

Figure 10 – Resource Processing and Dialog Adaptation

The Resource Processing step is responsible for DDL Dialog Template processing (see sect. 4.2).

The Resource Initialization component parses resource definitions in the Dialog Template. One resource instance is created for every resource definition found. Furthermore, the instance is bound to its DDL-Resource description and added to the resource instance set. Resource instances are managed as HTTP session objects. Thus, they are client specific and maintain their state among several HTTP requests. The Dynamic Content Inclusion component invokes resources for backend access referenced by the template's dynamic elements and replaces the enclosed field references in the template with backend data. Furthermore, this component handles the model state-based dialog content control described in sect. 4.4. At this point, all template sections have been processed.

Finally, the Document Processing step transforms the device independent representation into a target markup language through semantic and syntactic adaptation described in sect. 3.

5 Adaptation Framework

The Adaptation Framework (cf. 11) is the runtime environment of our adaptation system. The software design based on the "Chain of Responsibilities" [GaHe96] design pattern, thus enabling easy extension, implementation reuse and flexible reconfiguration even based on runtime conditions. Client requests are passed through a chain of filter components. The filter chain is executed in a Java Servlet.

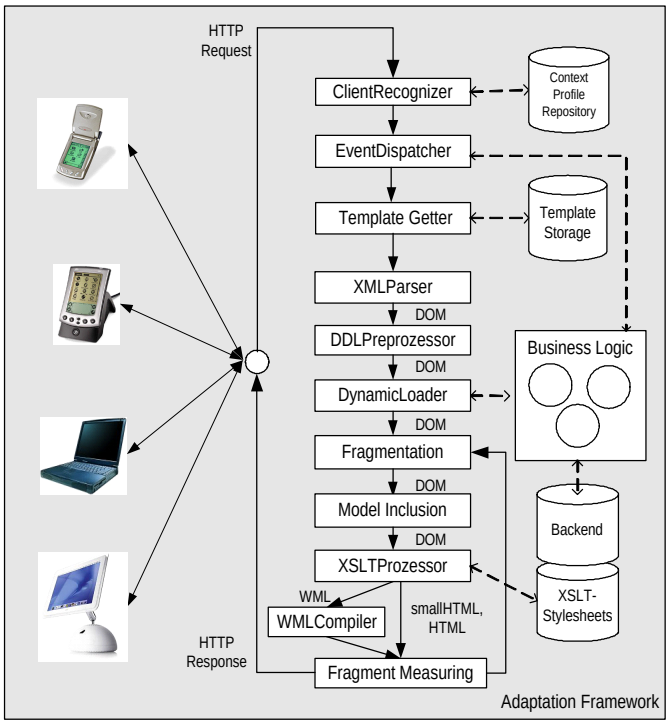

Figure 11: Filter chain of the Adaptations Framework

The implemented filters support our concepts for device independent event processing, resource processing and dialog adaptation. The system may be extended by implementing additional filters. The following section gives a brief overview of the filters and their functionality.

The *ClientRecognizer* filter maps the HTTP User Agent header received by the browser to a CC/PP [CCPP] device profile.

The *EventDispatcher* filter filters the events encoded in HTTP request and invokes the appropriate event handler resources (see sect. 4.3) utilizing a prioritized event queue to handle simultaneous occurrences of events.

The *TemplateGetter* filter retrieves DDL template according to the requested URI from the Template Storage.

The *XMLParser* filter parses the DDL document and transforms it into the DOM representation. The *PreprocessorFilter* preprocesses DDL dialogs. It is responsible for processing selective content exclusion and resolving inheritance hierarchies (see sect. 3).

The *DynamicLoader* filter implements the dynamic content processing described in sect 4.2. It is furthermore responsible for the interpretation of DDL resource descriptions and resource instantiation. Resource instances are bound to the HTTP-session between the client and the framework implementation.

The filters *Fragmentation* and *FragmentMeasuring* collaborate to perform the dialog fragmentation algorithm described in sect. 3.3. Fragmentation divides the dialog into atoms and reassembles the dialog according to the parameters determined for each atom by FragmentMeasuring. These parameters include memory size on the target platform, size in transfer encoding and screen area occupation.

The *ModelInclusion* filter performs the model-state dependent content control described in sect. 4.4.

The *XSLTProcessor* filter transforms DDL into a device specific mark-up through XSL Transformations.

The *WMLCompiler* compiles WML into the binary WMLC (WAP 1.x transfer encoding). Although this is a typical task of a WAP-Gateway, the WMLC code is required to probe the amount of space required by the dialog fragment within a WSP-Service Data Unit.

6 Sample Application

In order to evaluate the concepts presented in this paper, we have implemented the context-aware and adaptive Rent-A-Bike application shown in fig. 12. The application allows users to create, view, update, and delete reservations on desktop computers, PDAs and WAP mobile phones. All reservations and application-specific user profiles are stored in a relational database backend. The application logic and backend access is implemented by DDL Resources. User interaction is monitored by DDL events. Dynamic content, for example the ‘Your current reser-

vations' pull down menu list (Fig. 12(1) & 12(3)), is generated utilizing dynamic DDL. All versions of the application are generated from a single source DDL description. The HTML desktop version (Fig. 12(1)) is fully featured. It allows full access to the application's functionality and has the most complex user interface. The PDA version (HTML, Fig. 12(3)) has a reduced set of functions. It does not support reservation modifications. The user interface is less complex than on the desktop. In the PDA version, no icons are shown. The mobile phone version (WML, Fig. 12(2)) is even further reduced in terms of user interface complexity and application functionality. No headers, footers, and images are shown. There is no possibility to edit the user profile, i.e. the 'Profile' menu item is dropped. In contrast to the desktop version, reservations can be viewed, confirmed, and cancelled. There is no quick view option.

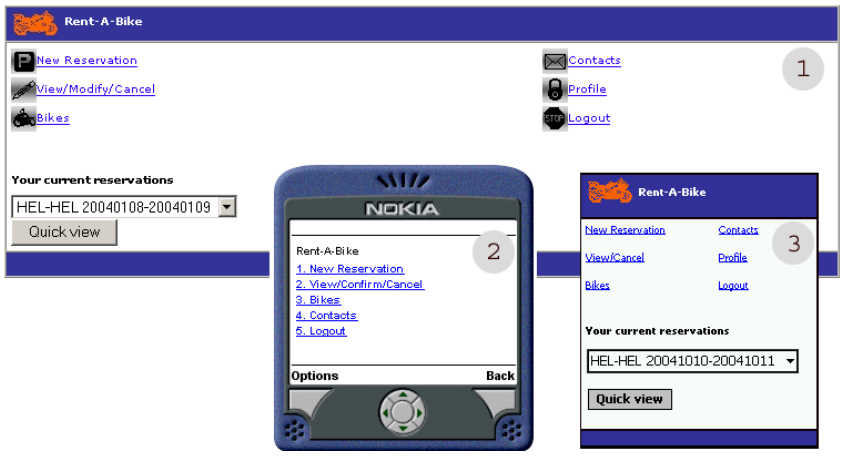

Figure 12 – Rent-A-Bike Application

7 Results and Future Work

We have found our approach, the combination of single source authoring with manually added meta-information and automated on-the-fly adaptation at runtime, to be a promising solution for the design and implementation of adaptive and context-aware web applications. It must be mentioned that application complexity in terms of user interface design and application logic behavior dramatically increases with the number of supported devices. This fact sometimes yields to a high amount of device-class specific DDL code, partly opposing the approach of highly

device-independent dialog authoring. To overcome this deficiency, we are currently developing advanced concepts for structure-level and application-level adaptation. Structure-level adaptation attempts to describe different navigation structures, dialog compositions and dialog flows. The dialogs necessary for this approach are assembled from semantic dialog units authored in DDL. Every semantic dialog unit describes a well-defined step or transaction within the application. Application-level adaptation aims at extending the description and definition of adaptive application behavior into the software-engineering process from requirements engineering over application design to implementation, test and deployment. Furthermore, performance analysis and enhancement will be subject to future work.

8 References

[AbHe00] Abrams, M., Helms, J.: User Interface Markup Language (UIML) v3.0 draft specification, Harmonia Inc, 2000.

[BiSc97] Bickmore, T.W., Schilit, B.N.: Digestor: Device-Independent Access to the World-Wide Web, Proceedings of the 6th International WWW Conference, 1997.

[Co04] Connexion by Boeing. http://www.connexionbyboeing.com

[GaHe96] E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns, ISBN: 0-201-63361-2, Addison-Wesley, 1996.

[HoKo00] Hori, M.; Kondoh, G.; Ono, K.; Hirose, S.; Singhal S.: Annotation-Based Web Content Transcoding, 9th International World Wide Web Conference (WWW9), Amsterdam, Niederlande, 15.-19. Mai 2000. http://www9.org/w9cdrom/169/169.html

[IDC00] IDC Research: Western European Teleworking: Mobile Workers and Telecommuters, 2000-2005 (IDC #25698)

[ScTr01] Schilit, B.N., Trevor, J, Hilbert, D., Koh, T.K.: m-Links: An Infrastructure for Very Small Internet Devices; Proceedings of the 7th Annual International Conference on Mobile Computing and Networking, Rome, Italy, pp. 122-131, 2001.

[W3C99a] W3C: XML Path Language (XPath) Version 1.0. 1999. http://www.w3c.org/TR/xpath

[W3C99b] W3C: XSL Transformations (XSLT) Version 1.0. 1999. http://www.w3.org/TR/xslt

[W3C03] W3C : XForms 1.0, W3C Recommendation 14 October 2003. http://www.w3.org/TR/2003/REC-xforms-20031014/

[WAP01] Wireless Session Protocol Specification, Approved Version 5-July-2001, Wireless Application Protcol Forum Ltd. http://www.openmobilealliance.org/tech/affiliates/wap/wap-230-wsp-20010705-a.pdf