Device Independent Representation of Web-based Dialogs and Contents

Steffen Göbel, Sven Buchholz, Thomas Ziegert, Alexander Schill
Department of Computer Science
Dresden University of Technology
D-01062 Dresden, Germany.
E-mail: {goebel, sb15, ziegert, schill}@rn.inf.tu-dresden.de

*Abstract*¾**Multi device service provision is a challenging problem for information service providers. To support the diverse set of existing and future devices services have to adapt the contents to the capabilities of the different devices.**

In this paper we present research efforts aiming at the development of system support for automated content adaptation to different devices. We especially target the generic description and automated adaptation of web-based dialogs.

We introduce an XML-based Dialog Description Language (DDL) enabling device independent description of dialogs. It makes up the basis for the automated adaptation. A fragmentation algorithm provides for automated decomposition of dialogs into dialog fragments that fit device specific resource limits. The feasability of our approach is proven by our Java-based adaptation framework, presented at the end of this paper.

*Keywords*¾**Multi device platform, Adaptation framework, XML.**

I. Introduction

The diversity of devices for mobile and stationary information access available today and expected for the future results in new challenges for web-based services. The heterogeneity in memory size, computing power and network connectivity as well as different content description languages must be taken into account by future information services. Unfortunately, the efforts and expenses involved in the manual adaptation of services to different devices are unreasonably high. Therefore system support for automated content adaptation is required.

Today's devices mainly support the following content description languages: HTML, WML, cHTML.

HTML has been developed for desktop computers and has evolved to different versions partly even with browser specific enhancements. Mobile devices, such as PDAs, mostly support a subset of the current HTML4 standard [1] only or limit its support to an older HTML version. Frames, Cascading Style Sheets (CSS) or active contents, such as JavaScript or Java, are fully supported by state-of-the-art desktop computer browsers, such as MS Internet Explorer or Netscape Navigator, only. The HTML subset supported by PDAs is also called tinyHTML due to its functional limitations.

WML [2] is an XML-based markup language. It is part of the WAP specification mainly supporting mobile phones or PDAs with small displays and restricted memory. Each WML page, also referred to as deck, may contain several cards. The browser supports navigation between those cards. Thereby server accesses are reduced.

cHTML has been deployed for the proprietary i-mode system [3] by NTT DoCoMo. Currently i-mode is solely deployed in Japan but there are plans to roll out i-mode service in Europe. cHTML is based on HTML 2.0 including some additional features. cHTML is not considered in our adaptation framework (cf. section V).

Automated content adaptation requires mapping the elements of a device independent content representation to the device specific content description languages. One approach to allow non-ambiguous mapping is restricting the features of the device independent markup language to those supported by all devices. The intersection of the features of all device specific markup languages makes the basis of the device independent format. With this intersection approach, the mapping to a device specific format is rather trivial because every feature of the device independent description can simply be translated into its device specific counterpart. However, a significant disadvantage of this approach is that the features of the device specific languages, and accordingly the capabilities of the devices, are not fully exploited. Furthermore, evolutions of a device specific markup language may need to trigger alterations to the device independent format.

As an alternative to the intersection approach a generic device independent format may support capabilities not featured by all device specific languages. Accordingly, the mappings of such a device independent language to device specific formats may be lossy because a complete and non-ambiguous mapping does not necessarily exist. The major advantage of such an approach is that the features of the generic language are not constrained by the device specific language with the least capabilities.

Each approach applying more than the intersection of device specific languages requires meta information about the semantics of language elements that cannot be mapped non-ambiguously. For example, meta information may indicate alternative representation of semantically one element. Furthermore, elements can

be declared to be optional and may be omitted on devices with insufficient resources. Just omitting or reducing contents can be very powerful for adapting to mobile devices with resource restrictions.

After a short survey about existing approaches for device independent description of dialogs (section II) this paper describes efforts in several fields of system support for automated content adaptation to different devices: a generic XML-based language (Dialog Description Language, DDL) for the device independent description of web-based dialogs (section III), a fragmentation algorithm to split complex dialogs into dialog fragments fitting the resource restrictions of the particular client (section IV) and a software architecture for automated adaptation of DDL dialogs to different devices (section V). The paper is concluded with an evaluation of our approach (section VI) and future directions (section VII).

II. Related Work

This section introduces existing solutions for device independent description of dialogs and discusses their advantages and disadvantages. This analysis was the starting point for the development of DDL.

A. *UIML*

The User Interface Markup Language [4] is an XML-based language for the description of complete user interfaces. Beside a header with meta information (for example author, creation time) a UIML document consists of: an interface description section, a peer information section, and an optional template section. The interface description contains information about the structure, style, content, and behavior of the user interface.

A great advantage of UIML is the strict separation of content, structure and style. By means of inheritance and inclusion of other UIML documents libraries of user interface components can be developed contributing to the reuse of code. Furthermore, UIML can be extended almost arbitrarily since UIML is a meta model.

UIML was designed for the modeling of complete user interfaces including mechanisms for event handling. Even parts of the application logic can be described in UIML. In consequence, even simple dialogs have a very complex UIML representation. This is a notable disadvantage of this approach. Furthermore, the currently available UIML renderers are not applicable for device independent description: Too many device dependent attributes are required forcing a developer to be aware of the different devices during the development process. Finally, UIML lacks of means to express constraints on valid user input.

B. *XForms*

XForms [5] are the W3C's response to the increasing demands for highly functional and portable web-based user interfaces. Just as UIML, XForms is built upon XML. It is intended to be integrated in future XML-based markup languages, such as successors of HTML, XHTML, SMIL etc. XForms is characterized by the separation of content, presentation, and logic, similar to UIML. The content is the fundamental data model, the presentation describes the appearance on different platforms, and the logic defines dependencies between form elements and additional functionality. This allows flexible presentation options (including classic XHTML forms) to be attached to an XML form definition.

XForms enables the validation of user input at the client within the browser. This reduces client-server interactions and server load. For that purpose XForms allows the specification of validity constraints on input data within the data model. Furthermore, XForms provides means to calculate values from user input at the client.

However, these XForms features require support by the browser software. This is not provided by the majority of today's browsers thereby preventing a rapid introduction of XForms. XForms is still under development and some parts have not been completely specified, yet.

C. *Oracle9i Application Server Wireless Edition*

The Oracle9i Application Server Wireless Edition [6] is a product supporting personalized services for different devices. However, it is a representative of the intersection approach (cf. section I).

Therefore the possibilities to design dialogs are very limited. Only a few elements have been defined (text, menus, forms, tables, containers, images). Besides the general disadvantages of the intersection approach there are some other drawbacks: There is no strict separation of content and style. Container elements cannot be nested complicating the structuring of dialogs. Tables are restricted to text and cannot held images, for example.

III. Dialog Description Language

Our experiences from the analysis of existing approaches for device independent description of dialogs have led to the definition of a new markup language unifying the identified required and desirable properties. The language is supposed to be simple and compact, but also functional, powerful and extensible. The Dialog Description Language (DDL), introduced in the following, fulfils these requirements. It follows the concept of decoupling of structure, representation and content, and the concept of inheritance. Constraints on user input can be specified. They allow automated input validation by the adaptation software.

DDL is an XML-based meta language. It describes a structure of abstract elements. Each of the elements can be assigned properties. In the following the syntax of DDL is described (cf. fig. 1). The root element of all DDL documents is <ddl>. A <ddl>-element may have the following descendants elements:

<include>: enables the integration of external DDL-source code into the current document. Therewith libraries of DDL code can be created and reused simply. By means of an optional "test"-attribute conditional inclusion can be realized. A condition is

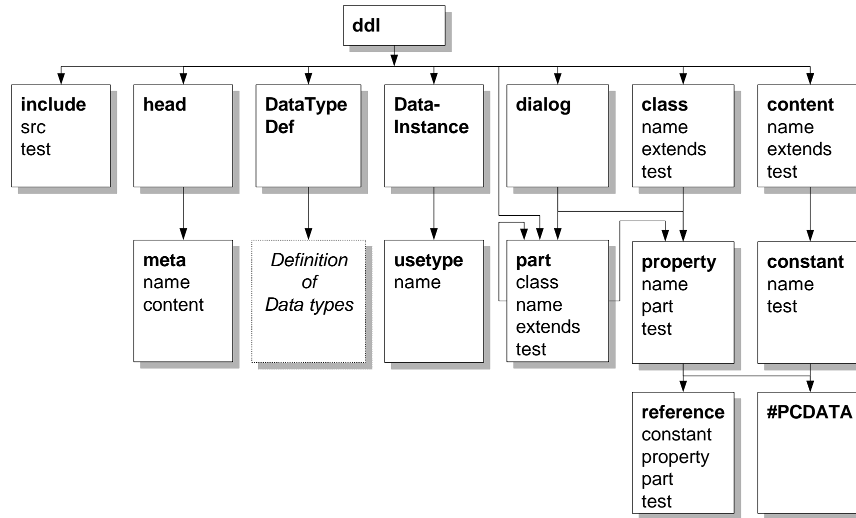

FIGURE 1
DDL SYNTAX

defined by an XPath-expression applied to the client profile.

<head>: enables the inclusion of additional meta information into the DDL document, e.g. author and creation date. This information is bracketed by a **<meta>**-element, a child element of <head>. There is at most one <head>-Element per DDL document.

<DataTypeDef>: defines data types used to validate user input in web forms. Each data type is assigned a unique name. Type inheritance is applied to express generalization and specialization relationships. There are basic data types (integer, float, date, string) as well as complex data types (structures, arrays).

Several restrictions (e.g. min and max value for integer data) can be assigned to each built-in basic data type allowing precise specification of valid inputs.

Type information can also be used to optimize the presentation of input elements. For instance, a calendar could be displayed to prompt for a date.

<DataInstance>: is used to specify data instances that are prompted for by the dialog. Each form input element has an associated data instance. By this means the user interface and the data is separated. Each <DataInstance>-element is assigned a unique name and a specific data type. The binding between a form input element and a data instance is realized via the data instance name.

<dialog>: starts the definition of the dialog structure. There is at most one <dialog>-Element per DDL document. It comprises all elements forming the actual dialog. Only those <part>-elements (see below) residing within the <dialog>-element are used for the output dialog. All other <part>-elements are library elements that can be used to inherit from.

<part>: <part>-elements are used to model the abstract structures of a dialog. They can be nested and may have properties (cf. <property>-element) assigned to them.

The optional "test"-attribute allows conditional inclusion of the part into the dialog analogously to the "test"-attribute of the <include>-element.

The optional "extends"-attribute refers to another part to inherit properties from. The referred part must be assigned a non-ambiguous name by a "name"-attribute. If properties are repeatedly specified in the inherited and the inheriting part, the properties of the inheriting part override the ones of the inherited part.

By means of an optional "class"-attribute a class (cf. <class>-element) can be assigned to a part. If a part belongs to a class and additionally inherits from another part, properties of the class have higher priority than those inherited from another part and therefore override them.

<class>: defines a class of parts. A class comprises a set of properties. All properties of a class are adopted by its instance parts. Analogously to parts, the optional "extends"-attribute can be used to define class inheritance.

By means of classes decoupling of structure and presentation is enabled. Perfect decoupling would mean that no properties are assigned directly to a part but to the class of the part only.

<property>: is used to assign properties (styles for the presentation or abstract properties) to parts or classes.

<content>: defines the contents of DDL dialogs. By means of <content>-elements DDL realizes decoupling of structure and contents.

The <content>-element comprises a set of data items (cf. <constant>-element) that can be referenced by <property>-elements or other <content>-elements.

<constant>: defines a data item as a literal (#PCDATA) or a reference (cf. <reference>-element).

<reference>: is a reference to a data item (<constant>) or property (<property>). It can be used as the value of a property or data item. A reference is expressed by the name of the referenced element as the value of the "constant"- or the "property"-attribute.

The semantics of the properties is defined separately and is not part of DDL. By this means future extensions may be developed without the need to change the

syntax (DTD) of DDL. Only the DDL renderers, for the adaptation to device specific languages, must be extended or adapted to be able to interpret new properties.

In the following we introduce a semantics for DDL to be applied by renderers for multi device dialogs.

Parts are assigned semantic types specifying the interpretation of the particular <part>-element by the renderer. Possible types and their properties are defined in the following (cf. fig. 2):

container – serves for grouping of arbitrary parts. By means of the "layout"-property the placement of the elements within the container is specified. Possible values are "horizontal", "vertical", "border" (analogous to the Java Layout Manager java.awt.BorderLayout), and "grid:*n*" (grid with *n* columns). The property "atom" labels indivisibly coherent elements. This property is interpreted by the fragmentation algorithm (cf. section IV). Atomic containers should contain as few elements as possible because a reasonable fragmentation is not feasible otherwise.

label – provides for presentation of text or links. By means of additional attributes the appearance (font, bold, italic) can be influenced.

image – enables the inclusion of images by specifying their URL. Devices that do not support images show an alternative textual description specified by the "alt"-attribute.

source – enables the inclusion of non-interpreted, device specific source code, e.g. HTML or WML. In conjunction with a "test"-attribute device specific code can be selected.

form – enables the definition of forms for user input. A form may have all properties of a container. After completing the form the user inputs are sent to the URL of the server ("action"-attribute) by the specified method ("method"-attributes) to be further processed by the server.

textinput – enables the input of text, values, and passwords within a form. Constraints on the input data can be defined by "match"-attributes, comprising regular expressions (similar to Perl5). Validation is executed at the server because a scripting language support at the client browser should not be presumed.

radiogroup – performs a one-out-of-many choice. Each option is placed by means of a **radiobutton-**part within the radiogroup.

checkbox – enables the input of a boolean value. The combination of several checkbox-parts allows for a many-out-of-many choice.

select – performs a single or multiple choice (depending on the attribute "multiple") by means of a selection list. Possible selections are defined by **option-**parts within the select part.

submit – shows a button to finish a form and submit it to the server. Forms without a submit part implicitly get a submit button labeled "OK".

frameset-parts and **frame**-parts can be used analogously to HTML frames.

In case the specified types are not sufficient for an application new part types can be easily added. Solely renderers (usually in the form of XSLT style sheets) have to be adapted.

The semantic type of a part is indicated by a <property>-element with the "name"-attribute="type". The type specific properties are assigned to parts by means of <property>-elements, too. For example, a link to another web site can be described by a label part in the following way:

```
<part>
   <property name="type">label</property>
   <property name="content">
      TU-Dresden
   </property>
   <property name="link">
      http://www.tu-dresden.de
   </property>
</part>
```

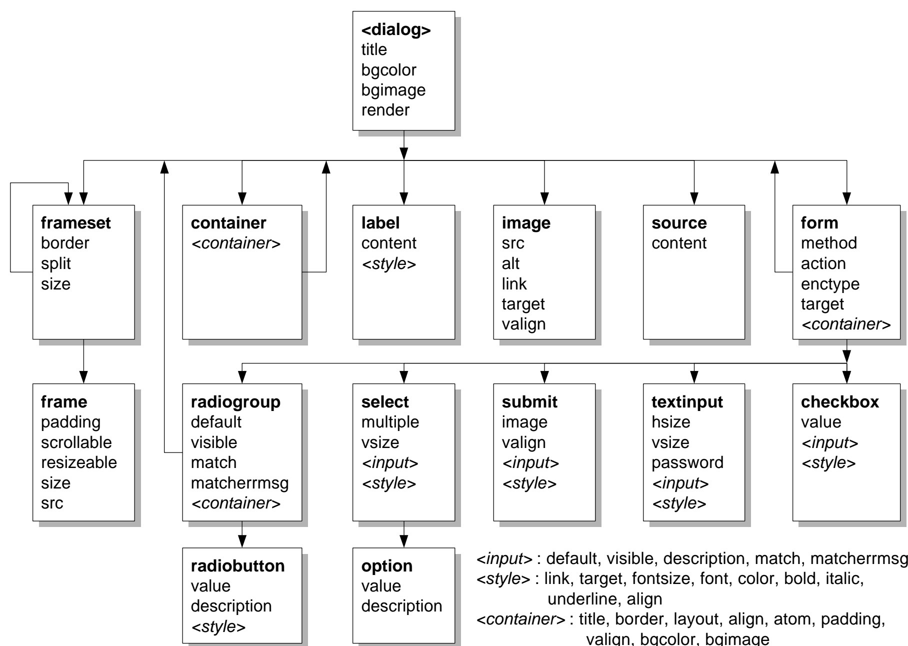

FIGURE 2
SEMANTICS OF DDL

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ddl SYSTEM "ddl.dtd">
<ddl>
  <dialog>
    <property name="title">User Login</property>
    <property name="bgcolor">FFFFFF</property>
    <part>
      <property name="type">form</property>
      <property name="method">GET</property>
      <property name="action">login.jsp</property>
      <property name="layout">vertical</property>
      <part class="container">
        <part class="label" name="nameLabel">
          <property name="content">User name:</property>
        </part>
        <part class="textinput" name="userInput"/>
      </part>
      <part class="container">
        <part class="label" name="passwordLabel">
          <property name="content">Password:</property>
        </part>
        <part class="textinput" name="passwordInput">
          <property name="password">true</property>
        </part>
      </part>
      <part class="okButton" name="okButton"/>
    </part>
  </dialog>
  <class name="container">
    <property name="type">container</property>
    <property name="atom">true</property>
    <property name="layout">horizontal</property>
  </class>
  <class name="label">
    <property name="type">label</property>
  </class>
  <class name="textinput">
    <property name="type">textinput</property>
  </class>
  <class name="okButton">
    <property name="type">submit</property>
    <property name="default">Ok</property>
  </class>
</ddl>
```

FIGURE 3

SAMPLE DDL DIALOG

An example for a complete simple login dialog is depicted in figure 3.

IV. FRAGMENTATION

Two major problems in presenting dialogs on multiple heterogeneous devices are differences in display and memory size. A desktop computer can usually display a complex dialog all at once but a PDA or WAP phone may require the fragmentation of the dialog.

Fragmentation must not be performed arbitrarily not to brake the logical structure of the dialog. Atomic parts of a dialog, that are placed in a container with the "atom"-property="true" (cf. section III), must not be partitioned by the fragmentation algorithm. For instance, an input element should not be separated from its description label.

Furthermore, the fragmentation algorithm should be as simple as possible to preserve computing power. We apply the following fragmentation heuristics:

1. A dialog is considered as a tree with container parts (atomic or non-atomic) as inner nodes and non-container parts as leaves.
2. Every leaf is assigned a specific weight indicating resource requirements (memory, display size etc.).
3. For every leaf that has an atomic container among its ancestors the node *A* is determined to be the one atomic ancestor that is the closest to the root (cf. fig. 4a).
4. All leaves with the same *A* node establish a group. Each leave without an atomic container among its ancestors is element of a group with a single element (cf. fig. 4b).
5. If there is just one group the dialog cannot be fragmented at all.
6. Dialogs with multiple groups can be fragmented in such a way that groups are not split and the sums of the leaves' weights within each fragment are balanced (cf. fig. 4c, for the sake of simplicity the example assume all leaves to have a weight of 1).

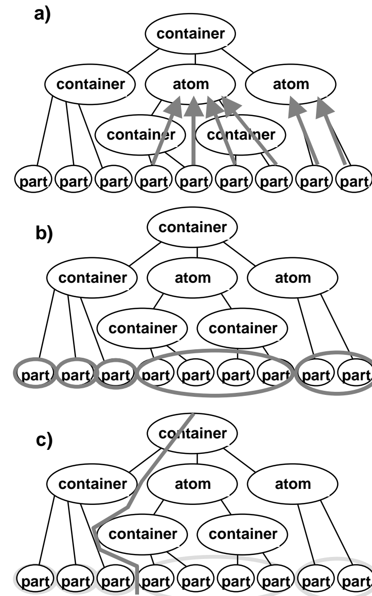

FIGURE 4

FRAGMENTATION ALGORITHM FOR DIALOGS

Currently we are further investigating optimization of the fragmentation algorithm. Especially the display size of a device should be considered more precisely. This includes taking image sizes into account.

Processing DDL dialogs implies some additional problems, that are solved by the adaptation framework introduced in the following section:

- Dialog fragmentation should be transparent to the application. To avoid inconsistencies a dynamic dialog has to be generated only once, not by fetching fragment by fragment.
- User input has to be cached until the whole dialog is completed and all dialog input is sent to the application at once.
- The user should be able to navigate between dialog fragments. Therefore appropriate navigation elements (e.g. links or buttons) have to be added to the fragments.
- If possible, user input should be validated fragment by fragment to allow immediate feedback and correction. In the other case the user must be prompted for correction after completing the dialog. In either case a feedback dialog is required to be generated automatically.

V. SOFTWARE ARCHITECTURE FOR ADAPTATION

To evaluate the applicability of DDL for the automated adaptation to different devices we developed an adaptation framework. It is based on Java servlets

and makes use of the Xalan-XSLT-processor [7] as well as the Xerces-XML-parser [8] by the Apache-Group.

The adaptation is performed via a chain of filters (fig. 5). The IBM WebSphere Transcoding Publisher [9] and the Brazil Project of Sun Microsystems [10] use similar architectures based on filters.

A filter is implemented as a Java class with a simple interface (TranscodingFilter) consisting of two methods: init() and start(). Additional filters can be implemented easily. We distinguish four different types of filters depending on their functionality:

- Request Modifiers – alter a HTTP request, e.g. include additional HTTP headers. Request Modifiers are processed at first.
- Generators – supply the requested contents. The simplest way to do this is to read a file from the hard disk. A more sophisticated solution may retrieve the contents from a DBMS, another servlet or an arbitrary URL. Basically there is exactly one Generator invoked during the processing of a single HTTP-request.
- Response Modifiers – adapt the retrieved contents to the capabilities of a particular device. They are linked in the filter chain after the Generator.
- Monitors – are primarily used to observe or debug an application and do not perform any modification. They may be linked at any position within the filter chain.

Filters are distinguished semantically only. The interface syntax of all filters is the same.

The sequence of filters in the request processing chain is determined by a configuration file. A single servlet running in the servlet container of a web server (e.g. Apache server with Tomcat) processes the configuration information and controls the successive execution of the filters.

The configuration file contains a mandatory sequence control attribute and an optional test attribute for each filter. The sequence control attribute sets the sequence of the filter invocation.

The test attribute, that contains an XPath expression about the client profile, provides conditional invocation of a particular filter. For instance, only requests from WML clients require the invocation of the WMLCompilerFilter (cf. fig. 5).

The transcoding process may include iterations, i.e. loop in the filter chain. To allow for loops a filter may optionally determine its successor. An example of a loop is the iterative fragmentation of a dialogs by DDLFragmentationFilter2 and DDLFragmentation-Filter3 (cf. fig. 5).

Currently we have implemented a number of filters empowering the adaptation framework to support the transformation of a DDL dialog into a device specific dialog representation. Additional functionality can be easily added by implementing extra filters.

In the following we give a brief overview about the filter classes and their functionality. All filter classes inherit some common functionality from the abstract superclass TranscodingFilterSupport and implement the common interface TranscodingFilter.

ClientRecognizerFilter (RequestModifier)

This filter is responsible for the client recognition and passes the device information as a client profile to the subsequent filters. In the current implementation device recognition is based on the User Agent String, which is part of each HTTP request. However, we are currently developing a subsequent version, that will use CC/PP [11] and a more general device classification. The adaptation process will rely on a particular device class rather than on a specific device. Hence, new devices will be supported without any modification of the system configuration.

URLGetterFilter (Generator)

This filter retrieves a file from the local file system or from the internet based on the URL in the HTTP request.

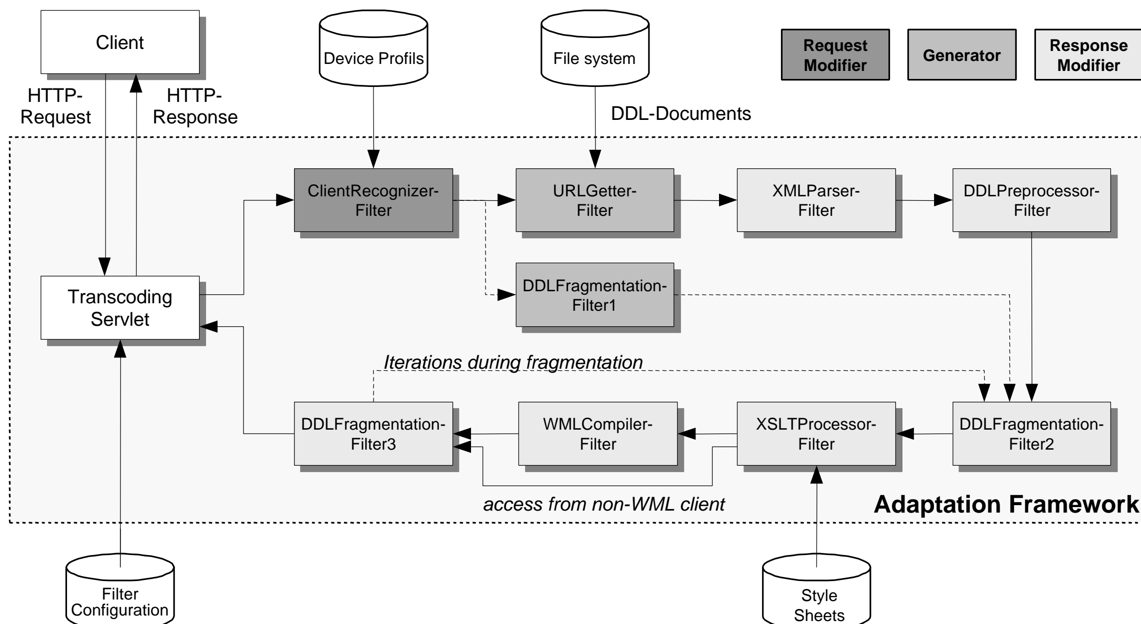

FIGURE 5

ADAPTATION FRAMEWORK

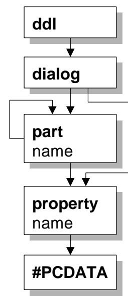

FIGURE 6

SIMPLIFIED DDL

ServletRunnerFilter (Generator)

This entity invokes an external servlet on the application server. By this means the system is able to generate dynamic contents in DDL.

XMLParserFilter (Response Modifier)

This filter converts an XML document (e.g. a DDL document) into an org.w3c.dom.Document object (Document Object Model, DOM [12]). A DTD can be specified to validate the document. Subsequent filters work on the DOM instance of the document.

DDLPreprocessorFilter (Response Modifier)

This filter preprocesses a DDL document to resolve all external references and inheritance hierarchies. This results in a simplified DDL document with single <dialog>-block (cf. fig. 6). By this means the style sheet-based transformation is eased. Even the preprocessing may be style sheet-based. However, as this process is very time consuming, we decided to use a DOM-based transformation.

XSLTProcessorFilter (Response Modifier)

This filter transforms a preprocessed DDL document into a device specific format (HTML, WML). The transformation is based on XSLT style sheets. The style sheets have access to the information in the HTTP request and the context of the processing environment. The information are presented to the style sheets as XSLT parameters.

There is a style sheet for each device (or device class). Currently we have implemented style sheets for HTML, tinyHTML and WML. The selection of the style sheet is based on the client profile generated by the ClientRecognizerFilter.

ImageTranscodingFilter (Response Modifier)

The image transcoder converts images, e.g. BMP into JPEG or WBMP, high resolution into low resolution, or full color into grayscale. Several format specific parameters can be specified, e.g. the "quality"-parameter for JPEG images or the "interlaced"-parameter for PNG images.

DDL fragmentation filters

These filters perform the dialog fragmentation in case the client's restrictions forbid the particular dialog to be displayed as a whole. Besides the actual fragmentation they are responsible for the user input validation and they store input data until the final dialog part is completed. The filters must appear in the following order:

DDLFragmentationFilter1 (Generator)

The first fragmentation filter manages the caching of dialog fragments and the fragment by fragment delivery to the clients. Furthermore it stores input data until the dialog is completed. Therefore input data is transferred to the filter by the query component of the URL of the HTTP request for the next dialog fragment.

FragmentationFilter2 (Response Modifier)

This component fragments dialogs by the fragmentation algorithm presented in section IV. Furthermore it validates user input.

FragmentationFilter3 (Response Modifier)

This component checks, if the size of the rendered document exceeds the resource restrictions of a particular device. If this is true, the filter invokes FragmentationFilter2 again to trigger another fragmentation.

WMLCompilerFilter (Response Modifier)

WAP devices do not process a textual WML document, but a compact binary representation (binary WML). Therefore a WAP gateway, an intermediary between the server and the WAP device, compiles the textual into the binary representation. However, this means the memory restrictions of the device do not apply to the size of the textual WML document but to the size of the compacted version.

To check if a WML document fits the resource restrictions of the client, a WMLCompilerFilter is interposed between the XSLTProcessorFilter and the FragmentationFilter3 to perform the conversion to binary WML.

VI. EVALUATION

To validate the functionality and applicability of DDL and our adaptation framework we have developed several sample documents. Figures 7 and 8 show a DDL sample dialog rendered on a WAP phone (WML)

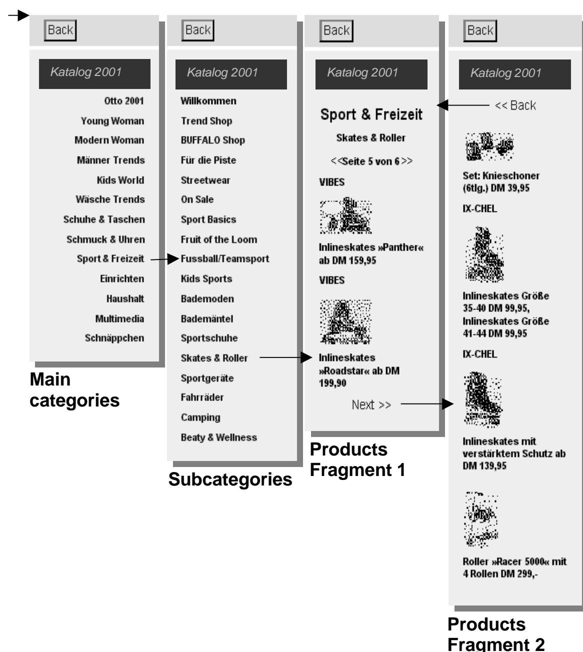

FIGURE 7

DDL DIALOG RENDERED ON A WAP PHONE

FIGURE 8

DDL DIALOG RENDERED ON A DESKTOP COMPUTER

and a desktop computer (HTML).

On a desktop computer there is just one frameset comprising three frames (fig. 8). On a WAP phone there are no frames but the frameset is split into separate dialogs (fig. 7). The entry point to the dialog flow is a list of main categories. The list of products is fragmented into two dialog fragments. A user can navigate between the fragments by means of “back” and “next” links inserted by the adaptation framework.

In order to evaluate our adaptation framework we have measured the performance for processing a complex DDL dialog. The total processing time is about 840 ms. Thereby the major share of the processing time is consumed by the XSLTProcessorFilter and the DDLPreprocessorFilter (cf. fig. 9).

VII. CONCLUSION

In this paper we have presented DDL, a device independent description language for web-based dialogs and contents. DDL is an extensible XML-based meta language that allows the description of dialog structures, content elements and constraints on user input. DDL documents are decomposable and reusable.

Furthermore we have introduced an adaptation framework that processes DDL descriptions to generate device specific dialogs.

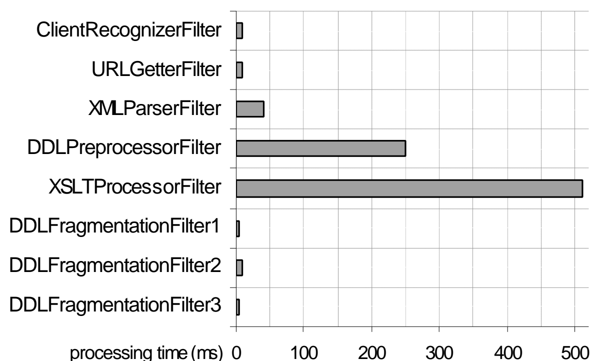

FIGURE 9

PROCESSING TIME FOR A COMPLEX DDL DIALOG

The evaluation of the framework has proven the XSLT-based transcoding performs rather poor. Therefore we are investigating optimization for the transcoding. Caching of transcoded contents is one conceivable approach. Another one may be to use compiled XSLT style sheets. This may lead to a significant speed-up and preserves the advantages of XSLT transcoding, such as flexibility.

In the future we plan to integrate personalization into the adaptation framework, e.g. personal settings regarding the adaptation process. Integrating those mechanisms into the existing architecture should be easily done by introducing new filters.

ACKNOWLEDGMENT

The authors want to acknowledge the contribution by Sebastian Weber to the design of DDL and the implementation of the adaptation framework.

REFERENCES

[1] D. Raggett, et al, *HTML 4.01 Specification*, W3C Recommendation, 1999. (http://www.w3.org/TR/html4/)

[2] *Wireless Application Protocol: Wireless Markup Language Specification 1.2*, WAP Forum Ltd., 1999. (http://www1.wapforum.org/tech/documents/SPEC-WML-19991104.pdf)

[3] *What's i-mode?*, NTT DoCoMo,Inc. (http://www.nttdocomo.com/i/service/home.html)

[4] *User Interface Markup Language (UIML) v2.0 Draft Specification*, Harmonia, Inc., 2000. (http://www.uiml.org/)

[5] M. Dubinko, et al, *XForms 1.0*, W3C Working Draft, 2001. (http://www.w3.org/TR/xforms/)

[6] *Oracle9iAS Wireless*, Oracle Corp. (http://www.oracle.com/ip/deploy/ias/index.html?wireless.html)

[7] *Xalan-Java version 2.2.D9*, The Apache Software Foundation. (http://xml.apache.org/xalan-j/)

[8] *Xerces Java Parser Readme*, The Apache Software Foundation. (http://xml.apache.org/xerces-j/)

[9] *WebSphere Transcoding Publisher*, IBM Corp. (http://www-4.ibm.com/software/webservers/transcoding/)

[10] *Brazil Project: Home*. Sun Microsystems, Inc. (http://www.sun.com/research/brazil/)

[11] G. Klyne, et al, *Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies*, W3C Working Draft, 2001. (http://www.w3.org/TR/CCPP-struct-vocab/)

[12] A. Le Hors, *Document Object Model (DOM) Level 2 Core Specification*, W3C Recommendation, 2000. (http://www.w3.org/TR/DOM-Level-2-Core/)