Adaptive Applications for Ubiquitous Collaboration in Mobile Environments

Allan Meng Krebs and Ivan Marsic
Center for Advanced Information Processing (CAIP)
Rutgers—The State University of New Jersey
Piscataway, NJ 08854-8088 USA
+1 732 445 4208
{krebs, marsic}@caip.rutgers.edu

Abstract

The wide availability of networked devices has become one of the main challenges for designers of collaborative applications. Mobile devices are not as “standardized” as stationary computers but much more diverse with respect to computing resources, connection bandwidth and display capabilities. This paper presents a framework for development of applications adaptive to the client’s computing platform. The framework supports adaptation of both shared data and user interface to user preferences and display characteristics. Shared data and the user interface are each specified by an XML document. The user interface XML document specifies the interface by a generic “interface graph”, which also includes the list of the valid data types. The generic interface graph is transformed into a device-dependent interface graph for individual client devices, using predefined device-specific mapping rules.

Keywords
Heterogeneous platforms, mobility, adaptive applications, ubiquitous computing.

1. Introduction

The old problem of adapting the application’s user interface to multiple platforms has become even more important with the large diversity of mobile connected devices that has emerged within the last couple of years. The use of various web browsers (HTML, cHTML, WML, etc.) offers some solution to the problem, but if the target application requires more complex features than is manageable using HTML forms, something else needs to be developed. Additional problems occur with collaborative applications, especially when the users are using heterogeneous computer devices. The shared data may be adapted and presented with different views depending on the computer platform used by a particular collaborator.

Platform independence is important for heterogeneous computing systems. Nowadays, most of the mobile handheld devices connect to standardized networks that are isolated from the Web. But more and more of them are starting to use wireless connections to corporate intranets and public Web services. Software downloading to handheld devices has been developed to leverage code mobility among mobile users. Again, the issues in user interfaces have been obstructing software portability. The demand for portable applications in the handheld device category has been increasing because these devices their networks are being used for new kinds of dynamic services. The diverse pool of mobile devices has brought entirely new requirements for portability and accessibility: the applications should be accessible from different computer platforms, working environments, and situations the mobile users are involved in. These developments introduced a strong emphasis on design and implementation of platform-independent graphical user interfaces for heterogeneous devices.

The kinds of applications that are commonly used in these scenarios mostly involve interactive data manipulation and browsing, unlike complex scientific computations. Thus, the application adaptability in this case translates to interface adaptability. The specific requirements that we identified for ubiquitous and adaptive collaborative interfaces in heterogeneous environments include:

- Device-independent application description that can be easily mapped to a device-specific implementation
- Dynamically adaptive user interface that adapts to contextual changes, such as changes in user interest or communication bandwidth
interface components, called interactors, in a
This is achieved as follows. We define a set of user
description into a device-specific implementation.
This is achieved as follows. We define a set of user
description of applications and a mapping of this
description into a device-specific implementation.

This paper addresses the problems of adapting the
application user interfaces to multiple platforms and
defines an infrastructure to support the development of
awareness tools to facilitate collaboration in heterogeneous
environments by specifying a language for generic
description of applications and a mapping of this
description into a device-specific implementation.

This is achieved as follows. We define a set of user
interface components, called interactors, in a data-centric
manner. The interactors are defined for the types of data
typically encountered in the targeted applications and thus
most of the interactors are application independent, i.e.,
generic. We define an XML-based markup language for
interactor specification. The interactors are then
implemented for each computer platform so that the
implementation is tailored to the platform capabilities.

Some of the interactors may not be supported on certain
platform since the platform may not be capable of
supporting such interaction. This forms the infrastructure
for developing adaptive applications.

The application developer uses the infrastructure to
develop new applications. A new application is generically
described in the interactor markup language. The developer
may need to program new derived interactors, but this
should happen very rarely. Our framework maps the
application description to the device-specific
implementation by associating the interactors with their
implementations. The mapping is influenced by the current
contextual information, such as network bandwidth or user
interests. If the context changes at runtime, the interface
may be re-mapped dynamically to reflect the changes.

The paper is organized as follows. We first review related
work. Next we describe the architecture of our adaptive
system and the interactor specification we developed for
defining generic application descriptions. Then we present
details on the mapping between the generic application
description and the device-specific application
descriptions, and the data flow through the system. Finally,
we discuss continuing work and conclude the paper.

2. Related Work

There are a few notable recent systems related with the
work described here. They usually address adapted user
interface generation, but not the dynamic modification of
it, and do not offer explicit support for heterogeneous
collaboration (e.g., XWeb, Visage, MECANO, BML,
M-Links, LiquidUI).

XWeb [[10]] proposes a method to define user interfaces
using the XView language. XView specifies a number of
standard user interface widgets called interactors that have
to be implemented on the different client platforms for
XWeb applications. XWeb mainly deals with string-based
interactors, including string representation of numbers, and
can generate html form-like views of the data. Our work
extends some of the ideas from XWeb focusing on user
interfaces for applications dealing with spatial data, such as
map-based applications. While generation of the view
using XWeb is a one-way transformation of the data using
a static view description, our approach supports change of
the view at run-time, both initiated by the user (or an
agent), or by changes to the generic application
description.

The Visage system from Maya Design [[8]] focuses on
complementary issues. Visage is a powerful (single-user)
visualization system for creating custom visualizations and
direct manipulation of large and diverse datasets. Some of
its unique features include dynamic data navigation
through drill-down and roll-up (aggregation) techniques.
Unlike our system does, while Visage addresses diverse
data visualization, it is not explicitly intended for
collaboration in heterogeneous computing environments.
Recent work on Visage Link adds multi-user collaboration
capabilities, but does not consider heterogeneous data
representations.

The early versions of MECANO [[12]] system (Stanford)
only used a domain model to automatically generate the
final user interface. In later versions, it provides its own
language (MIMIC) to define models of the type of users,
the tasks and goals of the user, the domain that the user can
affect by means of the interface, the presentation of the
interface, and the dialogue between user and interface.
These models are used by their own run-time system to
generate and control the final interface. The MIMIC
modeling language supports a design model that describes
dynamic relationships between entities of the other
declarative models.

Bean Markup Language (BML) [[2]] is an XML-based
component-configuration language for the JavaBeans
component model [[5]]. The language is directly
executable; i.e., processing a BML script will result in a
running application configured as described in the script.
BML can be used to describe the creation of new beans,
accessing of existing beans, configuration of beans by
setting or getting their properties and/or fields, binding of
events from some beans to other beans, and calling of
arbitrary methods in beans.

The m-Links system [[13],[14]] provides Web-browsing
support for small devices with limited text-based displays
(cell phones) by splitting the browsing activity into
separate modes: link navigation and content reading. The
system retrieves Web documents using the HTTP
The navigation interface exploits the user’s familiarity with
desktop file selection dialogs, after the links have been
analyzed, categorized, filtered, and new links have been
created to help the user navigate. The action interface
Finally, there are many systems that focus on code mobility (e.g., DACIA) without considering or giving a clean solution to the user interface adaptation problem.

DACIA ([6]) is a system that provides mechanisms for building groupware applications that adapt to available resources and support user mobility and dynamic reconfiguration. During execution, groupware applications components can be loaded, unloaded, or moved among hosts (with different computing power, memory or display capability), while maintaining communication connectivity with groupware services and other users. DACIA focuses mainly on mobility, intermittent connectivity, and application reconfiguration. It does not provide support to user interface specification and adaptation to the specific displaying device as our system does. An application user interface composed of complex graphical components cannot be migrated from a high-end display (of a PC) to a text-only display of a low-end device (such as PDA or mobile phone).

Although some components of the framework presented here may exist in other systems, to our best knowledge, there is currently no other research team that focuses on collaboration with heterogeneous platforms and addresses this broad range of issues.

3. Adaptive System Architecture

The system architecture is based on the DISCIPLE infrastructure middleware for collaborative applications in heterogeneous environments ([7]), which supports intelligent data distributing and transformation of data to heterogeneous platforms, but has no support for adaptation of user interfaces. Both the old and the new architecture are server based, with the server responsible for propagating updates of data between the clients and maintaining the global state of the system.

The new architecture comprises two parallel data flows, as showed in Figure 1. The server contains the description of the interface and the application data as two separate XML documents. The interface is expressed using interactors (defined below), which form the generic interface graph of the application interface. The generic interface graph is mapped into a device-specific interface graph, which is finally mapped into a Graphical User Interface widget tree. The widgets can be those supported by the language platform, such as Java Swing components, or they can be specially developed for this purpose.
On the other hand, the application data is represented as a collection of data objects in a repository (data graph). The data objects are encapsulated in uforms (short for “universal form”). A uform essentially consists of a unique identifier and a keyed map of properties. An example of an XML representation of a uform is shown in Listing 1, where it is seen that the type of the uform is CIRCLE and that it has three properties: XPOS, YPOS and RADIUS, which are all of the XML Schema type double.

The global repository of uforms is mapped into a device-specific local repository, which is then mapped into the interface graph and finally into the widget graph. In the last two steps, the individual uforms are mapped to corresponding interactors, and the widget implementations (Java classes) are loaded.

The application adaptation is done by mapping between the generic interface graph and the device-specific interface graph based on static device characteristics. In addition, our system is providing an interface for later support of dynamic adaptation to user preferences, context, and available communication and processing resources.

Programming new applications is essentially writing new generic application view documents, and if new data types are introduced, develop the necessary new interactor implementations to handle additional data types.

4. Interactors

The key entity of the generic application description is the interactor. Our definition of interactor is similar to the XWeb [10] definition, though our architecture using the interactors is different. An interactor is a specification of a view object that can handle specific data types. It does not specify input methods or layout; those are defined in a specific implementation. We define three categories of interactors: atomic, aggregate, and indirect.
4.1. Atomic Interactors

Atomic interactors define interaction on single data objects, such as numbers, dates, text, and enumeration of finite choices. These single data objects will in most cases be properties of a uform. As already stated, an interactor does not define the implementation of the user interface widget to handle the specific data type. The atomic interactor defines only which data type(s) are valid to be used with the interactor and an optional label. The atomic interactors we defined support a subset of the data types defined in the XML Schema [[15]]:

- Text (string)
- Numbers (float, double, integer and long)
- Primitives (boolean)
- Times (date and time)
- Enumerations (enumeration)

An example of the use of an atomic interactor can be seen in Listing 2, where a field to display and edit the balance are defined using the XML Schema data type double. Our long-term intention is to support all the XML Schema built-in data types.

Listing 2: Example the use of an atomic interactor

```
<Field name="balance" description="Balance">
  <data type="double"/>
</Field>
```

4.2. Aggregate Interactors

Aggregate interactors glue together multiple atomic interactors. The Group interactor is the simplest of the aggregate interactors, as it only specifies that the contained interactors should be logically presented together. We purposely do not want to specify geometric layout in the aggregate interactors as the device-specific implementation of the aggregate interactors can have very different layout capabilities. For example an audio-only interface have very different layout properties than a window based screen interface.

A special case of the Group interactor is the Application interactor that has the same properties as the Group interactor and also identifies the root of the generic application document. Only one Application interactor is allowed in an application description.

The Tree interactor can show and manipulate an arbitrary number of data objects (uforms) in a tree structure. The Tree interactor can for example be implemented as a tree view of the objects, but could also be implemented in another way as the Tree interactor only specifies that the data objects handled is structured in a tree.

Like with the Tree interactor, the SpatialGroup can manipulate an arbitrary number of uforms, but the SpatialGroup interactor is targeted to manipulate spatial data, such as objects on a map. The SpatialGroup interactor can contain the views of an arbitrary number of uforms. The view can be implemented as two- or three-dimensional. The interactor description defines the data objects the interactor can handle.

The ChoiceGroup interactor is a group interactor that selects the contained interactors depending on the type of data object to be handled. In order to use a ChoiceGroup interactor, the developer defines a number of data objects to handle, and the appropriate interactors for each data object. An example of a definition of a ChoiceGroup interactor can be seen in Listing 3. The ref attribute of the ChoiceGroup is used to reference the data object to be handled.

Listing 3: Example of a ChoiceGroup interactor.
```
<ChoiceGroup name="Figure Editor" ref="id(…)">
  <Choose type="Rectangle">
    <ChoiceGroup name="Figure Editor" ref="id(…)">
      <Field type="double" name="radius"/>
    </ChooseGroup>
    <Field type="double" name="width"/>
    <Field type="double" name="height"/>
  </Choose>
</ChooseGroup>
```

4.3. Indirect Interactors

The indirect interactors do not contain objects they but are manipulating objects in other interactors, e.g., in a Tree or SpatialGroup interactor. The only indirect interactor we have defined is the Tool interactor that is used to define interface tools for an application. The definition of a Tool interactor specifies what data types the tool can manipulate, a reference to the interactor that contains the data objects, and what the tool does.

5. Mapping Interactors to Widgets

The server maintains the generic description of the application interface as a generic interface graph (see example in Listing 4). This generic interface graph can be pre-loaded into the server or a client can request the loading by providing a URL to the document containing the description. The server maintains a list of active or pre-loaded generic interface graphs, which can be requested by the user for selection of the application.

To create the user interface the generic interface graph needs to be mapped to the device-dependent representation. The former one is expressed in the interactor language defined above and the latter is
expressed in terms of a interface graph that refers to device-dependent GUI widgets.

This mapping is not necessarily a one-to-one mapping. For example, if the general description defines a 3D scene, this will be mapped to a single 3D view on a workstation, but it can be mapped to several views on a 2D device (top, side and front views).

The rules used for the mapping are device specific, but not application specific, i.e. the mapping rules are defined so that for each device the same rules are used for all application descriptions. The rules are specified as XSL [[16]] template rules and are only applied as a complete XSL transformation of the generic application description the first time a new device specific description is needed. If the generic application description is changed later the rules are only applied to the changes, and the necessary changes are merged into the device specific description.

**Listing 4: Generic application description for the sample application.**

```xml
<Application name="SlowTetris">
  <Group name="Tools">
    <Tool type="rotate" name="Rotate z">
      <ref>../../../SpatialGroup[@name="&amp;#34;View&amp;#34;]"></ref>
      <data type="polygon"/>
      <property name="axis" value="z"/>
      <property name="angle" value="90"/>
    </Tool>
    <Tool type="rotate" name="Rotate y">
      <ref>../../../SpatialGroup[@name="&amp;#34;View&amp;#34;]"></ref>
      <data type="polygon"/>
      <property name="axis" value="y"/>
      <property name="angle" value="-90"/>
    </Tool>
    <Tool type="fit" name="Fit">
      <ref>../../../SpatialGroup[@name="&amp;#34;View&amp;#34;]"></ref>
      <data type="polygon"/>
    </Tool>
  </Group>
  <Application name="SlowTetris">
    <SpatialGroup>
      <data type="document"/>
      <data type="overlay"/>
      <data type="polygon"/>
      <SpatialGroup name="View" dimension1="x" dimension2="y" dimension3="z">
        <data type="polygon"/>
      </SpatialGroup>
    </SpatialGroup>
  </Application>
</Application>
```

The XSL template rules can be generated either manually (with a standard text editor) or by a rule editor to simplify the process for the user.

The result is a interface graph, defining the structure of the GUI. This is the model of the application view (user interface). An example can be seen in Listing 5. This model defines the view of the application, which is implemented as a tree of widgets (Java code). Both model and view of the user interface are maintained to enable persistence of the user interface model and to enable other clients to subscribe to updates on the user interface model.

The final assembly of the user interface can either be completed on the client platform or on the server. In the first case the server will provide the device specific interface graph to the client, which then can load the necessary Java classes from local storage or download them from a remote class repository. In the latter case, the server will provide the complete binary code to the client, as for example a Java MIDlet.

Not all the attributes of the device specific interface graph can be defined at the time of transformation as for example screen layout and size are not necessarily know at that time. It is the responsibility of the implemented widgets and layout managers to update the device specific interface graph with the necessary attributes, when the user interface is laid out. For example the position of a frame implementing a group interactor cannot be determined at transformation time, but has to be computed by the implemented layout manager.

### 6. Mapping Data to Views

A collection of uforms represents a data graph that roughly corresponds to an XML document. As described above, each interactor defines the set of data types it can handle. The atomic interactors mainly used to handle properties of a uform, and will typically be placed together in a group or a choice group to allow editing of a uform. In this case the mapping between data and the view is done by the group is being associated with a given uform using a reference to the data structure. The atomic interactors get associated with the corresponding properties.

For example, implementing a property editor, which can edit the properties of a uform the user selects, can be done by having a special uform associated with each user, which includes a selected property field that refers to the selected uform. The group interactor handling the selected uform will have a reference to the selected property field and if it is not empty, associate with the uform referenced by it. If the user can select multiple types of uforms, the choice group interactor will be used to choose between different sets of atomic interactors. The group interactor can be used, if only one type of uforms can be selected.

The aggregate interactor types Tree and SpatialGroup are defined to handle a collection of uforms, and present them either as a tree view or as a spatial view (e.g. on a map). The implementations of these aggregate interactors are associated with the uforms by a mechanism where they get attached to the local repository containing the uforms, and the repository will then provide them with references to the uforms of types they have listed as valid. The repository will also provide notification upon addition or deletion of uforms.
The actual visualization of the data handled by the Tree and SpatialGroup interactors is totally dependent on the implementation. A typical implementation of the Tree interactor in Java would be using the javax.swing.JTree class, while for SpatialGroup the Java 2D API or Java 3D API could be used for the implementation, dependent on the target platform.

Listing 5: Interface graph description for 3D view of the sample application.

```xml
<Application ID="200000" name="SlowTetris">
  <Layout ID="200001" class="cWorld.WinLayout">
    <View ID="200002" class="..." title="Tools">
      <Panel ID="200003" class="...">
        <Component ID="200004" name="Rotate z" ...>
          <property name="target" type="[L">
            <item value="200008"/>
          </property>
          <property name="types" type="[String">
            <item value="polygon"/>
          </property>
          <property name="axis" type="String" value="z"/>
          <property name="angle" type="Double" value="90"/>
        </Component>
        <Component ID="200005" name="Rotate y" ...>
          <property name="target" type="[L">
            <item value="200008"/>
          </property>
          <property name="types" type="[String">
            <item value="polygon"/>
          </property>
        </Component>
        <Component ID="200006" class="...">
          <property name="target" type="[L">
            <item value="200008"/>
          </property>
          <property name="types" type="[String">
            <item value="polygon"/>
          </property>
        </Component>
      </Panel>
    </View>
    <View ID="200007" class="..." title="View">
      <Panel ID="200008" class="...">
        <property name="typemap" type="java.util.Hashtable">
          <item name="document" value="cWorld.glyphs.Document"/>
          <item name="overlay" value="cWorld.glyphs.Overlay"/>
          <item name="polygon" value="cWorld.glyphs.Polygon"/>
        </property>
      </Panel>
    </View>
  </Layout>
</Application>
```

7. Interactive Updating of Interface graphs

The device specific interface graph can be updated both by the client actions and by modifying the generic application interface graph.

After the initial transformation of the generic interface graph into a device specific interface graph, a mapping of the elements is maintained between the two interface graphs. This mapping enables the transformation of later changes to the generic interface graph to be applied only to elements, which was updated, and the whole device specific interface graph is therefore not overwritten. Only if the basic structure of the generic interface graph is changed a complete transformation will be applied.

To propagate the interface graph changes to user interface, the client subscribes to be notified on changes, using the same mechanism as the subscription to changes in the data graph.

A key advantage of this architecture is that it provides an infrastructure to support awareness. Other clients can subscribe to changes to a specific client’s interface graph. This can be used to provide awareness tools, like shared views, radar view or telepointers.

Also the users can update the generic interface graph. User can add extra collaboration components to a running application, for example a chat window. This is done by first adding the desired component to the device specific interface graph, then mapping the specific interface graph to the generic interface graph and propagating it back to the device specific interface graphs of the other users, as shown in Figure 2.

The server maintains a uform containing a list of the active users with links to the device specific interface graphs. This supports user interfaces enabling the users to browse the list of active users, and to subscribe to updates to the device specific interface graph of a selected user.

8. Interactive Updating of Data Graphs

Updating of the data graphs contained in the global and local repositories is done using a set of commands. The defined commands includes commands for adding, updating, and deleting whole uforms, adding, updating, and deleting properties of uforms, and some commands to maintain the client/server relationship.

When a user wants to change a data object, an update command is sent from the client application to the server to update the global repository. A command sent to update their local repository will then notify other users subscribing to changes to the global repository. A data adaptation agent handles the decision on what changes to notify the subscribing users about. The user can provide a set of rules to the data adaptation agent that defines the objects the user are interested in and under which circumstances the user wants notification about changes. For example, the user can set up the rules to receive updates on object changes depending on the available network bandwidth. The rules can also be set up to aggregate the data to send only updates about higher-level
objects in a hierarchical data structure. The interactions between the client and server components in the process of data adaptation can be seen in Figure 3.

9. Implementation and Example Applications

Presently we have only implemented sample widgets for the Java 2 Standard Edition (Java 2 SE) platform, but we are working on implementations for the Java 2 Micro Edition (Java 2 ME) Personal Profile and the Java 2 ME Mobile Information Device Profile (MIDP) platforms. Using the framework presented here we developed two complex applications for the Java 2 SE platform: a 2D graphics editor (Pocketscape) and a 3D virtual world (cWorld).

9.1. cWorld

cWorld is an application that allows users to do multi-user, synchronous, collaborative work in Collaborative Virtual Environments (CVE’s). It is developed using the Java 2 SDK v.1.3.1.0_02 and the Java 3D 1.2.1 API OpenGL version. cWorld does not require any special hardware and can be used with a keyboard and a mouse or employ more specialized devices, such as the Magellan Space Mouse.

cWorld allows the users to directly manipulate objects in the CVE, e.g., rotate or move, by providing a set of manipulation tools.

For the purpose of a user study on heterogeneous collaboration, which this specific version of the cWorld application was used for, we built a constrained version of the full cWorld. In this version, cWorld provides three predetermined views to facilitate the movement in the collaborative environment. This modification was introduced to alleviate in part the difficulties in object alignment encountered in the previous study. We expect that these changes will reduce the variability introduced by the user’s level of ability needed to use the mouse in a 3D environment.

The user can rotate an object 90 degrees clockwise around the γ-axis, 90 degrees clockwise around the z-axis, and fit it into the wall. The object rotation and placement in the wall do not depend on the use of the mouse. These functions are simplified using interface buttons that rotate the 3D scene objects 90 degrees in the y-axis (yaw) and z-axis (roll) and fit them in the wall once they are in the correct position.

9.2. Pocketscape

Pocketscape application was developed using Java 2 SE. Its GUI is implemented using the Swing toolkit. We implemented Pocketscape because we needed a user interface that could fit on the small screen of a PocketPC (320x240 pixels). Pocketscape runs on a Compaq iPAQ PocketPC 3536 handheld with 32 MB of memory and running the SavaJe XE operating system. SavaJe XE, unlike Windows CE, offers a full featured Java Virtual Machine that supports JDK 2.

The user input is captured via the handheld stylus and the PocketPC buttons. The user can log in to a collaborative session, load a 3D scene, log off the collaborative session, or exit the application. It features customizable tools like select, create, delete, move, and rotate objects. Both the tool set and the 3D scene are declared as XML files and loaded at the application startup.

Similarly to cWorld, Pocketscape is also modified to load a 3D scene and show its 2D front and top views (Figure 4). These modifications allow the subjects on PDAs to toggle between a top view and a side view using the stylus. Along the same lines, the rotation and placement of the objects is done by clicking buttons in a tool bar. Rotation of the
objects is done by selecting the rotate tool (once), selecting the rotation plane (top or front view), and touching the object with the stylus. For each touch the object rotates 90 degrees clockwise in the rotation plane. When an object is in the position that allows it to fit with the object on its left side, fitting is done by selecting the fit tool, selecting a view (front or top view), and touching the object with stylus. There is no scrolling in the interface.

9.3. Discussion of Code Reusability

An important benefit of this architecture is in code reusability resulting from using the same interactor implementations in different applications. For the above-described applications (cWorld and Pocketscape) the only interactor that is unlikely to be used in many other applications is the implementation of the SpatialGroup interactor. This interactor contains about 30% of the code in the cWorld implementation and about 25% in the Pocketscape implementation. We believe that this trend will continue for other application specific interactors, such as interactors for spreadsheet or calendar applications.

10. Conclusion and Future Work

We have designed and built a system that allows for adaptation of the user interface to multiple devices for collaborative applications. We have an indication that the code reuse when building new applications using our architecture will be high (possibly 70-75%). We have extended the idea of interactors to support arbitrary data set, with a focus on spatial data, and defined a mapping between a generic application description and device specific application descriptions. The data flow of application descriptions is defined so both changes of the local user interface done by the user and changes of the generic application description can be handled.

We are currently focusing on extending the list of interactors to include most of the XML Schema built-in data types, to implement clients for the Java 2 Micro Edition (Java 2 ME) Personal Profile and the Java 2 ME Mobile Information Device Profile (MIDP) platforms, and add adaptation based on task modeling by dynamically changing the mapping rules.

11. References


groupware applications using a mobile component
framework. CSCW 2000, December 2-6, 2000,
Philadelphia, PA, USA. ACM, 2000, pp. 107-116

groupware applications. Proc. 23rd IEEE/ACM Int’l
Conf. on Software Engineering (ICSE 2001), Toronto,

http://www.maya.com/visage/base/technical.html

[9] Mozilla Open Source Web browser,

[10] Olsen, D. R., Jefferies, S., Nielsen, T., Moyes, W., and
Fredrickson, P., Cross-modal Interaction using XWeb,
Proceedings of UIST ’00, San Diago, CA, USA,

Applications with XUL, WebTechniques, vol. 5, issue
7, (July 2000).

and Integrated Support for Model-Based Interface
Development, Computer-Aided Design of User
Interfaces, ed. by Jean Vanderdonckt, Presses
Universitaires de Namur, Namur, Belgium, pp. 19-25,
1996.

Links: An Infrastructure for Very Small Internet
Devices. In Proceedings of the 7th Annual
International Conference on Mobile Computing and
Networking 2001 (Rome, Italy, July 16-21 2001),

T. K. m-Links: From desktop to phonetop: A UI for
web interaction on very small devices. In Proceedings
of the 14th Annual Symposium on User Interface
Software and Technology (UIST'01).

Recommendation 02 May 2001.

[16] XSL Transformations (XSLT) Version 1.0, W3C