

Mobile Adaptive Applications for Ubiquitous Collaboration in Heterogeneous Environments

Allan Meng Krebs, Ivan Marsic, and Bogdan Dorohonceanu

Center for Advanced Information Processing (CAIP)

Rutgers University

Piscataway, NJ 08854-8088 USA

+1 732 445 4208

{krebs,marsic,dbogdan}@caip.rutgers.edu

ABSTRACT

Mobile teamwork applications are often deployed in heterogeneous environments with various devices and connections. To support a broad variety of platforms for mobile teamwork in heterogeneous environments, this paper presents a framework for development of applications adaptive to the client's computing platform. The framework supports both shared data adaptation and user interface adaptation to user's preferences and display characteristics. Both shared data and the user interface are specified in two XML documents. The user interface XML document specifies the application interface by a generic "view" graph (should also specify the valid data types). Global view graph is transformed into a device-dependant view graph for individual client devices.

Keywords

Heterogeneous platforms, adaptive applications, ubiquitous computing.

INTRODUCTION

The old problem of adapting applications user interfaces 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, WAP, etc.) gives some solution to the problem, but if the needed application is much more complicated than manageable using HTML forms, something else needs to be developed.

Other problems occur when the applications become collaborative, especially when the users are using heterogeneous devices. Besides developing a system which can handle sharing of the same data, but presenting it with different views, also the human factors of heterogeneous collaboration plays a big role. One way to facilitate the collaboration in heterogeneous environments is the use of awareness tools, which helps the user to be aware of what the peers can see and are doing.

The research presented in this paper addresses the problems of adapting application user interfaces to multiple platforms, and at the same time defines an infrastructure to support 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 description. The descriptions are dynamic and can be updated by the users at run-time.

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 further work and conclude the paper.

RELATED WORK

The system described in this paper is very related to XWeb [1], which defines a method to define user interfaces using the XView language. XView specifies a number of standard user interface widgets called interactors, which has 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 [4] 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. 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 models or data.

Although some components of the framework presented here are seen to exist in other systems, to our best

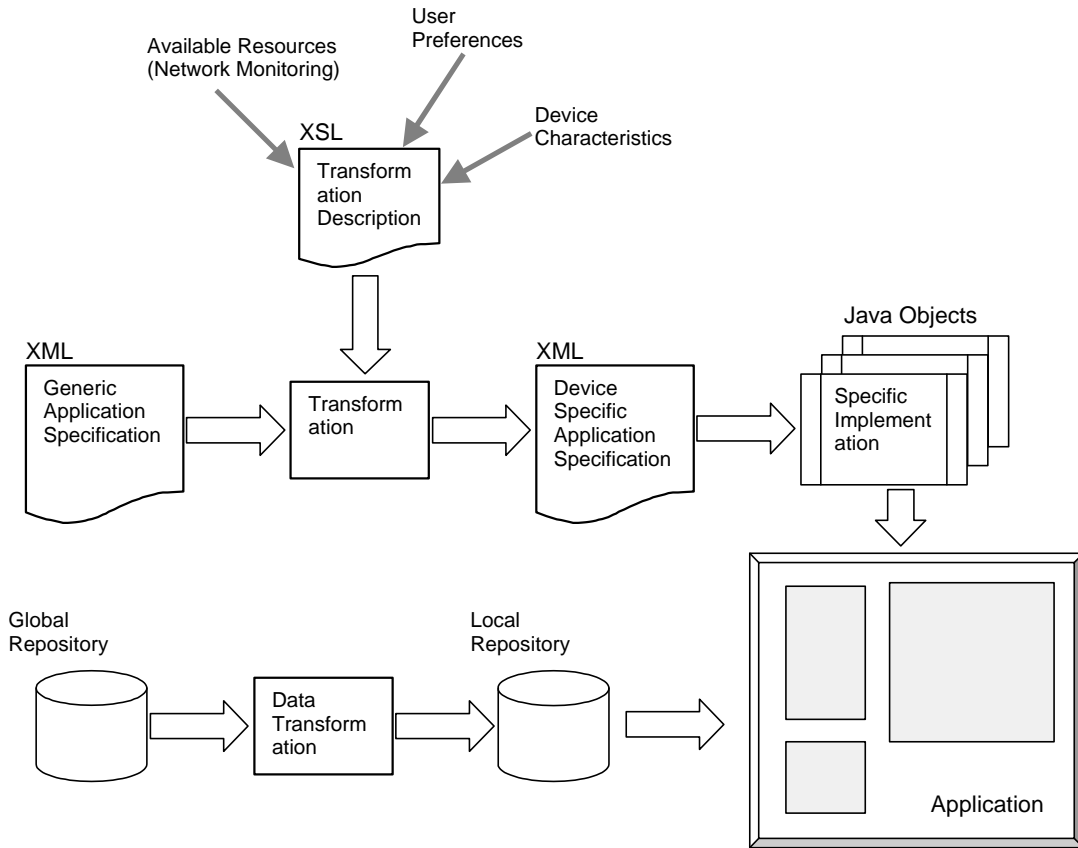


Figure 1: User interface data flow (top row) and application data flow (bottom row) result in the application.

knowledge, there is currently no other research team that focuses on collaboration with heterogeneous platforms and addresses this broad range of issues.

The Situated Computing Framework (SCF) [7] focuses on combining the personalization and ubiquity nature of handheld devices with powerful capabilities of static devices. The goal is to let the PDA dynamically create a composite device by seamlessly exploiting surrounding computing resources like PCs, storage devices, TVs, or telephones. The assumption is that screen size will remain rather constant (due to requirement to be pocket-sized) while small devices will move towards higher computing power, screens towards higher resolution, wireless networks will provide higher bandwidth, and multimedia content and services will move towards mobile-centric. The focus is on architectural consideration (logical-view) and much less user interface issues. The design goals are: detection of available location-based composite devices, ability to organize, arrange, manage, synchronize multimedia content in order to ensure service deliveries to output clients, rich content interactivity based on small screen units, redirection of server response to most appropriate composite device in vicinity of user, permit server to invoke processes on output device to deliver services.

The early versions of MECANO [8] 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 interface. The MIMIC modeling language supports a design model that describes dynamic relationships between entities of the other declarative models.

The UIML [12] project is in many ways related to our approach, but is not targeted towards collaborative applications and do not provide the infrastructure to do dynamic changes to the user interface as is included in our approach. UIML are also intended to be used on very low-level entities of the user interface e.g. defining frames and menu items, while our system is intended to be used on higher level components.

The MOTION [13] web-based architecture for peer-to-peer collaboration also have some relation to our system, but besides being only a peer-to-peer system, it also only deals with user interface adaptation using browser technologies.

ADAPTIVE SYSTEM ARCHITECTURE

The architecture supports two parallel data flows, Figure 1. The server contains the description of the application interface and the application data as two separate XML documents. The interface is expressed using interactors

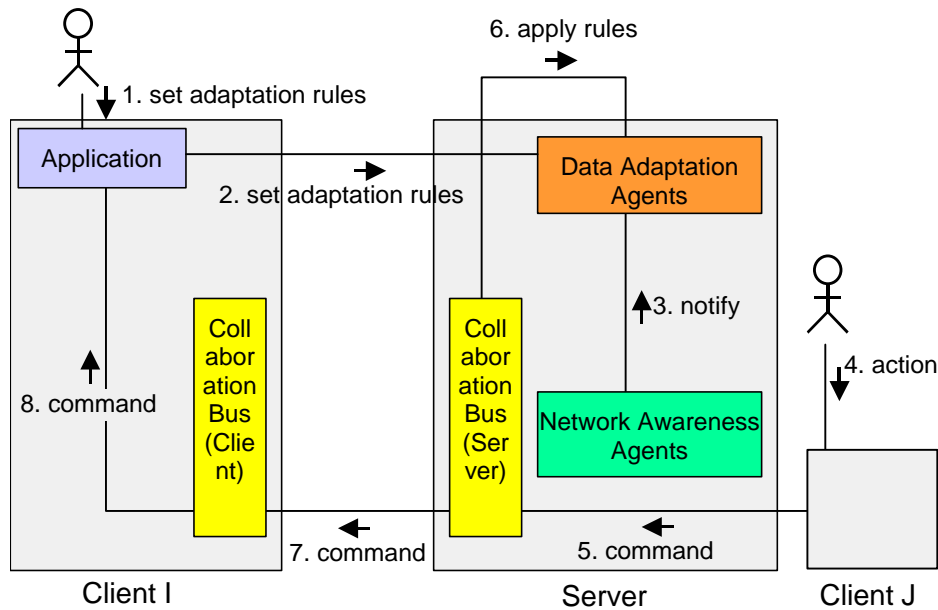


Figure 2: The interactions between the framework components in the process of data adaptation to the context.

(defined below), which form the generic *view graph* of the application interface. The generic view graph is mapped into a device-specific view 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 are represented as a collection of data objects in a repository (data graph). The data objects are called uforms (short for “universal form”), which encapsulates the data. The uform essentially consists of a unique identifier and a keyed list of properties¹.

The global repository of uforms is mapped into a device-specific local repository, which is then mapped into the view 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 we describe below is the adaptation of applications to a variety of client computer platforms. The adaptation is done by mapping between the generic application view graph and the device-specific view graph based on static device characteristics. In addition, our system is designed to also support dynamic adaptation to user’s preferences and to available communication and processing resources, Figure 2.

Programming new applications is essentially writing new generic application view documents.

¹ The concept of UForms is borrowed from [4], which in turn derives from e-forms [5].

INTERACTORS

The key entity of the generic application description is the interactors. Our definition of interactors is similar to the XWeb [1] definition. An interactor is a specification of a view object, which can handle specific data types. It does not specify input methods or layout; this is defined in a specific implementation. We define three categories of interactors; atomic, aggregate and indirect.

Atomic Interactors

Atomic interactors define interaction on single data objects, such as numbers, dates, text, and enumeration of finite choices. 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 [2]:

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

Our long-term intention is to support all the XML Schema built-in data types.

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. A special case of the *Group* interactor is the *Application* interactor, which has the same properties as the *Group* interactor, plus it also identifies the root of the generic

```

<ChooseGroup>
  <Choose type="Circle">
    <Field type="double" name="radius"/>
  </Choose>
  <Choose type="Rectangle">
    <Field type="double" name="width"/>
    <Field type="double" name="height"/>
  </Choose>
</ChooseGroup>

```

Figure 3 - Example of a ChoiceGroup interactor

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 in a tree structure.

The *ChoiceGroup* interactor is a group interactor, which selects the contained interactors depending on the type of data object to be handled. 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 Figure 3.

A special case of the aggregate interactors is the composite interactor. A composite interactor can contain other interactors like the group and list interactors, but is targeted to manipulate spatial data, such as objects on a map. E.g. *SpatialGroup* is a group interactor that can contain an arbitrary number of interactors for graphic symbols. The view can be implemented as two- or three-dimensional. The interactor description defines the data objects the interactor can handle.

Indirect Interactors

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

MAPPING INTERACTORS TO WIDGETS

The general description of the application interface (see example in Figure 4) needs to be mapped into the device dependent representation. The former one is expressed in the interactor language defined above and the latter is expressed in terms of a view 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).

```

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

```

Figure 4 - Generic application description for the sample application

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 [3] template rules, but are only applied as a complete XSL transformation of the generic application description the first time a new device dependent 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 dependent description.

The XSL template rules can be made, either by hand in a standard text editor, or by a rule editor to simplify the process for the user.

The result is a view tree, defining the structure of the GUI. This is the model of the application view (user interface). An example can be seen in Figure 5. In addition the view of the application is the tree of widgets (Java code).

MAPPING DATA TO VIEWS

A collection of uforms represents a *data graph*, which roughly corresponds to an XML document. As described above, each interactor defines the set of data types it can handle. Given an XML document, the composite interactor creates other composite or leaf interactors and associates them with corresponding uforms. It is important to notice that we maintain both model and view(s) components for each uform. This is unlike XWeb and for this reason we do

not require one-to-one mapping from a model to a view. In addition, the data views are not described by interactors as in XWeb. The user interface interactor directly maps the data to GUI components, thus bypassing the interactors.

INTERACTIVE UPDATING OF VIEW GRAPHS

The device specific view graph can be updated both by the client and by changing the generic application view graph .

The client subscribes to be notified on changes to the view graph, using the same mechanism as the subscription to changes in the data graph. Also other clients can subscribe to changes to a specific clients view graph. This can be used to provide awareness tools, like shared views, radar view and telepointers.

Also the users can update the generic application view 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 view graph, which is then mapped to the generic application view graph, and at the end propagated to the other users.

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.

IMPLEMENTATION AND EXAMPLE APPLICATIONS

Using the framework presented here we developed two complex applications: a 2D graphics editor (Pocketscape) and a 3D virtual world (cWorld).

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 only with a keyboard and a mouse, but it can also 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 our user study, we built a constrained version of 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 [9]. 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 y-axis, 90 degrees clockwise around the z-axis, and fit it into the wall. The object's rotation and placement in the wall does not depend on the use of the mouse. These functions are simplified using buttons in the interface that rotate the blocklets 90 degrees in the y-axis (yaw) and z-axis (roll) and fit them in the wall once they are in the correct position.

Pocketscape

Pocketscape application was developed using Java 2 SE. Its GUI is implemented using the Swing toolkit. It is a simpler version of the Flatscape editor and has the basic

```
<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>
```

Figure 5 - View graph description for 3D view of the sample application

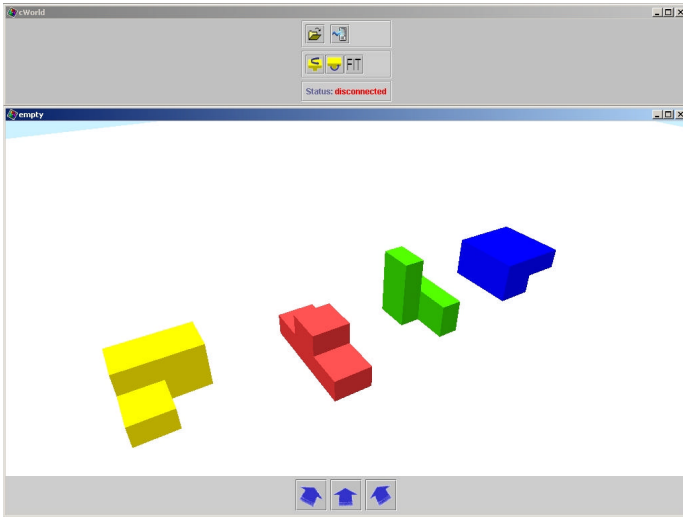


Figure 6: A configuration of 3D objects built using cWorld. Shown is the left view of the configuration.

functionality of Flatscape. We implemented Pocketscape since 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 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, e.g., 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 7). 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.

CONCLUSION AND FUTURE WORK

We have designed and built a system, which allow for adaptation of the user interface to multiple devices for collaborative applications. 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

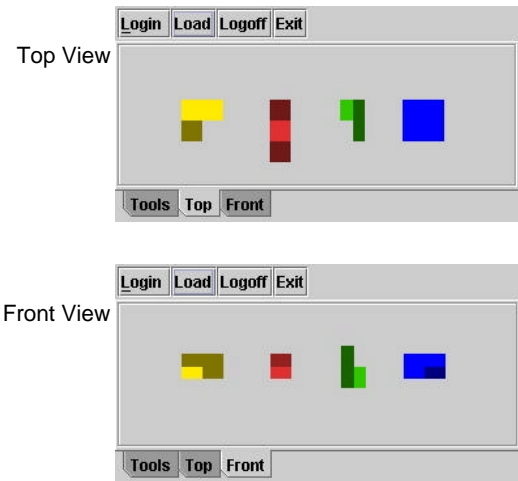


Figure 7: A view of the parts as seen using Pocketscape, corresponding to the configuration shown in Figure 2.

the user, and changes of the generic application description can be handled.

In the future we hope to extend the list of interactors to include most of the XML Schema built-in data types, and add adaptation based on task modeling by dynamically changing the mapping rules.

REFERENCES

1. Olsen, D. R., Jefferies, S., Nielsen, T., Moyes, W., and Fredrickson, P., Cross-modal Interaction using XWeb, *Proceedings of UIST '00*, San Diego, CA, USA, November, 2000, pp.191-200
2. XML Schema Part 2: Datatypes, W3C Recommendation 02 May 2001
3. XSL Transformations (XSLT) Version 1.0, W3C Recommendation 16 November 1999
4. Maya Design Group, Inc. Visage Link. <http://www.maya.com/visage/base/technical.html>
5. Dertousos, M. *What Will Be*. New York: Harper Collins, 1997, pp.85
6. Marsic, I. An architecture for heterogeneous groupware applications. *Proc. 23rd IEEE/ACM Int'l Conf. on Software Engineering (ICSE 2001)*, Toronto, Canada, May 2001, pp.475-484
7. Thai-Lai Pham, Georg Schneider, Stuart Goose: A situated computing framework for mobile and ubiquitous multimedia access using small screen and composite devices, *ACM Multimedia 2000*, Los Angeles, CA, USA, October/November, 2000, pp.323-331
8. Puerta, A. R. The Mecano Project: Comprehensive and Integrated Support for Model-Based Interface Development, *Computer-Aided Design of User Interfaces*, ed. by Jean Vanderdonckt, Presses Universitaires de Namur, Namur, Belgium, 1996, pp. 19-25

9. Marsic, I., Krebs, A., Dorohonceanu, B., and Tremaine, M. Designing and examining PC to Palm collaboration. *Proc. 35th Hawaiian Int'l Conf. on System Sciences (HICSS-35)*, Waikoloa, Hawaii, January 2002
10. John L. Schnase, Edward L. Cunnius, S. Bruce Dowton: The StudySpace Project: Collaborative Hypermedia in Nomadic Computing Environments, *Communications of the ACM*, Volume 38, Number 8, August 1995, pp.72-73.
11. Radu Litiu, Atul Parakash. Developing adaptive groupware applications using a mobile component framework. *CSCW 2000*, December 2-6, 2000, Philadelphia, PA, USA. ACM, 2000, pp. 107-116
12. Marc Abrams, Constantinos Phanouriou, Alan L. Batongbacal, Stephen M. Williams, Jonathan E. Shuster, UIML: An Appliance-Independent XML User Interface Language, *The Eighth International World Wide Web Conference*, Toronto, Canada, May 11-14, 1999, <http://www8.org/w8-papers/5b-hypertext-media/uiml/uiml.html>.
13. Gerald Reif, Engin Kirda, Harald Gall, Gian Pietro Picco, Gianpaolo Cugola, and Pascal Fenkam A Web-based peer-to-peer architecture for collaborative nomadic working. In Proceedings of the 3rd International Workshop on Web-based Infrastructures and Coordination Architectures for Collaborative Enterprises, co-located with the 10th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2001), MIT, Cambridge (MA, USA), June 2001., <http://www.elet.polimi.it/Users/DEI/Sections/Compeng/Gianpaolo.Cugola/Papers/wetice2001.pdf>.