MIMA: Multimedia Interfaces for Mobile Applications

Enrico Bertini, Giuseppe Santucci, Tiziana Catarci
Dipartimento di Informatica e Sistemistica - Università di Roma "La Sapienza"
Via Salaria, 113 - 00198 Roma, Italy - {bertini, santucci, catarci}@dis.uniroma1.it

***Abstract*— We are living in a "digital age", digital libraries, digital cameras, digital TVs and there is, therefore, the need to support the library users to fluidly, continuously, and seamlessly interact with the digital entities, and operate within the digital and physical settings. This need is particularly true in the context of mobile computing, where device limitations, in terms of screen dimension, input capabilities, bandwidth, and computational power, require new techniques and tool to pursue effectively such a goal. In the above framework, the MIMA systems intends to address the main problems associated with the customizable transmission of video streams on portable devices, such as PDAs and smart phones. To this aim, the project deals with several strictly interrelated aspects, allowing for automatically constructing video summaries, designing in a declarative way the user interface, and adapting the interface presentation to the actual portable device. In this paper we focuses on the interface design and adaptation.**

I. Introduction

In this paper we deal with the problem of designing user interfaces for multimedia applications that can run on multiple heterogeneous devices while ensuring usability. In order to address this issue, the next generation of multimedia internet applications should be designed for adapting to a very large spectrum of different characteristics. Currently, the research is faced with the problem of finding models and techniques to make applications aware of and adaptable to: (1) devices, e.g., cellphones, PDAs, PCs, kiosks, (2) environments, e.g. noisy room, low light, moving person, and (3) users, i.e., many different users in terms of preferences and capabilities.

In this paper we focus on the first issue, i.e., adapting the application to different, mobile devices, taking into account audio-visual issues and we deal with three different topics: (1) architectures that support heterogeneous devices, (2) models to exchange and share data, and (3) techniques for building effective user interfaces.

Our approach is mainly based on the idea of modeling the user interaction in very abstract and simple way. It is in our believes that, to deal with the many different implementations a single application must support, it is fundamental to have a single abstract model able to define the user interaction. Having an abstract layer permits, in fact, to decouple the activity of defining the dialog of the service from the activity of implementing the service for multiple different contexts.

The paper is structured as follows: in Section II we describe some related proposals, in Section III we introduce a model that supports the design of adaptive interfaces for audio-visual applications: the *Atomic Interaction Units* model and a working example is provided. Section IV, provides a description of the main device characteristics we want to take into account. Section V describes some preliminary results on the `AIUs` implementation strategies, and finally in Section VI some conclusions and open issues are discussed.

II. Related works

The problem of generating different interfaces for different devices having a single common application, is often indicated as the problem of creating *plastic interfaces* [8], that is, create user interfaces that gently change their form according to devices characteristics. All the related proposals share the common idea of using formal models and well structured processes, in which the generation of the final interface is just the last step of a composite activity.

Puerta and Eisenstein in [2] propose a design framework that consists of a *platform model*, a *presentation model*, and a *task model*. The designer builds the user interface by means of abstract interaction objects that are platform-neutral widgets he can assemble to design the interface in an abstract manner. With a similar approach, in [6] the Teresa tools is presented. It provides an environment to design multi-device applications that is strongly based on task modeling. The interesting aspect of this proposal is that the presentation is automatically generated from a task model.

The same idea of heavily exploiting formal models to design interactive applications comes from research on data-intensive web design, as illustrated in [3], that stems from past research on model-based hypermedia design, like RMM [5] and HDM [4], and that has a major focus on data modeling. This approach suggests a process in which the designer starts from a model of the data (usually drawing an entity-relationship model) and on top of it creates an hypertext model that defines the connection points between the web pages, i.e., the links. WebML [1] is a powerful data-driven language that permits to describe an hypertext composed of single atomic blocks that are tightly connected to underlying data elements and that can be connected in order to pass parameters through them. In WebML it is possible to design, upon a single common data model, different hypertext structures (*site views*), that are automatically selected in order to serve the most appropriate according to the characteristics of the connected device. Unfortunately, it is not possible to aid the designer into the construction of different site views for different devices.

Our work shows similarities with all these systems, in fact here we propose a model-based approach. We adopt the idea

of providing a collection of atomic interaction units that are the abstract counterpart of common interaction elements. At the same time our work diverges from this approach and comes closer to data-intensive web modeling approaches. Hence, we propose to directly design the hypertext and, thus, the structure of links that connect the presentation units. As it will be described in detail below, our method fundamentally consists in specifying a graph structure (a UML Activity Diagram) in which the nodes are populated with atomic interaction units and the edges represent the transition triggered by the termination of such interaction units.

III. Formalizing the interaction: the Atomic Interaction Units

The foundation of our proposal is an abstract model able to characterize the user interaction, modeling the information that is exchanged between the user and the system together with the purpose for which such an information is exchanged. Using this approach the designer is provided with a formalism to specify the information content of each presentation and the connection between the various parts, in order to indicate the behavior of the application, that is, how the system evolves as the user interacts with it. Our proposal consists of two main parts:

- A set of Abstract Interaction Units (`AIUs`) to be used as building blocks for abstract interface definition
- The UML *Activity Diagram* as formalism to connect the `AIUs` that compose the interface

The set of `AIUs` has been produced analyzing the user interfaces that are actually used to model standard web services together with suitable primitives able to dial with audio-video applications. Starting from specific interaction elements, we have grouped them into higher level units based on functional similarity. Such units express the key interactive features the specific elements of each group have in common. The challenge is in collecting a small set of atomic units that could describe the interaction, abstract enough to be completely unrelated with the particular device on which the interface must be realized, but expressive enough to let designers model complex services. The effort we made has produced a small set of `AIUs` .

The UML *Activity Diagram* is basically a state chart diagram in which each state represents an activity and each transition is triggered by the end of this activity. The model, as can be argued, seems to be quite appropriate to describe the elements we are dealing with. Moreover, considering that UML has become the standard de facto for designing applications, we can avoid to build an entirely new model. How the *Activity Diagram* can be used to glue together the `AIUs` will be explained later by using an example.

A. The set of AIUs

We foresee two main interaction activities: browsing, i.e., just observing something produced by the system and inputting, i.e., providing the system with some information; each `AIU` is characterized by a signature that defines the input it expects and the output it returns. It is out of the scope of this paper to present the complete list of implemented `AIUs` ; in the following we discuss some details of a subset of them, to give a more concrete feeling of our approach. Note that all the `AIUs` share a Quit command that allows for leaving the `AIU` with no effects and returning the null value.

```
BrowseVideo
(VideoId, VideoDescription,
VideoResolution,
ListOfBrowsingCommands):
{NULL, elemOfListOfBrowsingCommands}
```

The `BrowseVideo AIU` allows for browsing a video stream; usual facilities of play, pause, stop are provided, if possible, by the device. The video description is a two values record: [`VideoName`, `VideoSummary`], where `VideoName` is used as a title during the image presentation and `VideoSummary` is an image description that can be used when the video channel is not available or disturbed. The `VideoSummary` can be used considering the environment or the user as well. As an example, assume the user is driving a car or is visully impaired; in such a case, if the audio channel is available the interface can exploit it to deliver the `VideoSummary`. The `ListOfBrowsingCommands` is a set of commands oriented towards server side image manipulation (e.g., changing video resolution, moving to the next video extracted characteristics, etc.). The null value signals that the user terminated the `AIU` through the Quit command.

USAGE: the purpose of this `AIU` is to allow the user to explore some pieces of information presented in a graphical way.

```
BrowseTable
(TableId, TableDescription,
ListOfBrowsingCommands):
{NULL, elemOfListOfBrowsingCommands}
```

The `BrowseTable AIU` allows for browsing a relational table (with or without BLOBs and CLOBs); usual facilities of scrolling are provided, if possible, by the device. The table description is a two value record: [`TableName`, `TableSummary`], where `TableName` is used as a title during the table presentation and `TableSummary` is a text description that can be used when the video channel is not available or disturbed or as an alternative when the device capability of displaying a large table is very poor. The `ListOfBrowsingCommands` is a set of commands oriented towards server side table manipulation (e.g., moving quickly to a tuple); such commands do not allow to reach any other state than the one hosting the `AIU` (i.e., they correspond to self-transition). The null value signals that the user terminated the `AIU` through the Quit command.

USAGE: the purpose of this `AIU` is to allow the user to explore some pieces of information presented in a tabular way.

```
InteractTable
(TableId, TableDescription,
ListOfBrowsingCommands, Mode):
{NULL,elemOfListOfBrowsingCommands,tableTuple)
```

The `InteractTable AIU` is quite similar to the `BrowseTable AIU` . The main difference is that the user can leave the `AIU` by selecting a tuple (that is returned by the `AIU`).

USAGE: this `AIU` allows the user for providing the system with some input, i.e., the user selects a tuple from a table.

B. AIUs at work

So far we have seen how elementary interaction activities are modelled with `AIUs` . Now we want to describe how the composition of these units can lead to the design of a whole service. The

UML Activity Diagram is used to compose the AIUs and define the service. Each activity state can contain one AIU and models the activity of the user with the specific interaction unit. That is, the activity state is the abstraction of the atomic interaction with the system (e.g., choosing an element in a list). The transition between one state to the next is triggered by the user acting with the specific interaction unit and each transition correspond to a computation operated on the server. The interaction units that return some output can instantiate a variable that can be used in guard conditions to conduct to different states.

In order to clarify the use of this model we provide an example describing how a simple service to select a video fragment can be modeled. We will refer to Figure 2 that depicts the activity diagram filled with the specific AIUs utilized to model such a service. The figure comes from the working prototype we implemented, MaisDesigner, that allows for modeling the user interaction and generating the interfaces for different devices.

The user starts selecting an interesting video among a list through an *InteractTable* AIU . The implementation of the *InteractTable* is shown in the topmost part of Figure 2(b). Once the video has been selected (Lazio-Roma football game, in the example), the user is presented with a picture that shows, visually, the automatic or manual annotations available for that video. This activity is modeled through an *InteractImage* AIU ; the implementation of is visible in the center of Figure 2(b) and contains three kinds of annotations: normal play, goals, and throw in. In the example the user is interested in goals and selects the first red item. The system activate a *BrowseVideo* AIU , whose implementation is visible in the bottommost part of Figure 2(b).

After browsing the video, the user is asked through a *SelectChoice* to select one of three alternatives: 1) browse again the same video fragment, 2) select a new segment in the current video stream, 3) select a new video (through the Home button).

This simple example shows how the composition of the abstract interaction units allows for modeling a service. After this phase, the system must be able to translate this model into a final implementation. In order to perform the translation the system must take into account a set of key device characteristics. Hence, in the following section, we describe the characteristics that we consider necessary to perform the adaptation.

IV. Devices and AIUs metrics

In order to effectively implement the AIUs on physical devices, we need some figures about their capabilities. Different classifications and characteristics are available in the literature (e.g., [9]). Here, we focus on a first set of characteristics that constitute the minimal information needed to adapt the different AIUs to each device. Moreover, we need to investigate the AIUs as well because of the size of the parameters they handle heavily affects their implementation (e.g., the way in which the user interacts with a relational table may differs depending on the number of tuples and attributes). A practical usage of such parameters is shown in Section V. Again, a full description of the metrics used for characterizing devices and AIUs is out the scope of this paper; for reference purpose we provide just few examples of them.

A. Devices oriented metrics

Concerning devices we use, among the others, the following functions:

- int RN(dev) (Row Number), returning the number of rows the device is able to display;
- boolean CVS(dev) (Continuous Vertical Scrolling), returning the availability of a continuous (i.e., pixel based) vertical scrolling;
- boolean JE(dev) (Java Enabled), true if the device is Java enabled;

B. Aius oriented metrics

Concerning the AIUs , we shows some metrics about the text based AIUs , distinguishing between table and pure text oriented AIUs .

- int RN(table-oriented-AIU), (Row Number) returning the number of rows the AIU needs to be displayed;
- int CN(table-oriented-AIU), (Column Number) returning the number of columns the AIU needs to be displayed;
- int CHN(text-oriented-AIU), (CHaracter Number) returning the number of characters the AIU needs to be displayed.

V. Issues on AIUs implementation

So far, we have seen how through the AIU model an interface can be designed in abstract. This is the first phase of the whole process. In fact, after having designed the system, a final implementation must be generated. As already mentioned, various approaches can be followed, static and dynamic. Here we propose a dynamic adaptation based on the metrics described in the previous section.

A systematic analysis of the AIUs implementation is out of the scope of this paper. Here we discuss, as a working example, the implementation on the device d of the AIU a, InteractTable used in the example shown of Figure 2. Assume that, among the others, the following figures hold for the involved device and AIU :

- RN(a)= 40 (i.e., the table needs 40 rows);
- CN(a)= 75 (i.e., the table needs 75 columns to be displayed);
- RN(d)= 20 (i.e., the device can handle 20 rows);
- CN(d)= 30 (i.e., the device can handle 80 columns);
- RVS(d)= false (i.e., the device does not allow for row based vertical scrolling);

Based on these figures, we can argue that the AIU can be easily displayed for what concerns the number of columns; on the other hand, handling 40 rows on a device that is able to display only 20 rows and does not allows for horizontal scrolling is not immediate. The only way is to present the table to the user in a multi steps interaction:

1) the user is presented with a table containing only a subset of the table tuples whose occupation is less than 20 rows;
2) additional commands allow for getting all the available videos, simulating the vertical scroll.

As an example, we can see in Figure 2 a possible implementation of the InteractTable AIU on a device with reduced capability device (the image is presented on a Nokia 9500 device, reducing, for test purpose, the number of available rows to 20 and simulating the non availability of the vertical scroll). In such a table only four video at time are available and two new commands (up and down) simulate the vertical scroll.

The above example provides the feeling on how it is possible to adapt the same AIU to different devices. We are currently investigating different implementation strategies and different threshold values to come up with a more formal and complete way of implementing AIUs .

The actual implementation includes a design environment, i.e., MaisDesigner, that allows for drawing the UML activity diagram modeling the service we are dealing with, and a server side implementation, using servlets and running under Tomcat.

VI. Future work and conclusions

In this paper we presented a novel approach for implementing, on a variety of portable devices, audio-video browsing applications. The main ideas supporting our proposal are: (a) the availability of a formal model characterizing the user interaction (AIUs and UML), (b) the formalization of the characteristics of devices and AIUs through several suitable metrics, and (c) the definition of ad-hoc strategies for implementing in efficient way the AIUs on different devices.

Some aspects of our approach deserve more deep analysis: we are currently working on defining a complete set of AIUs and

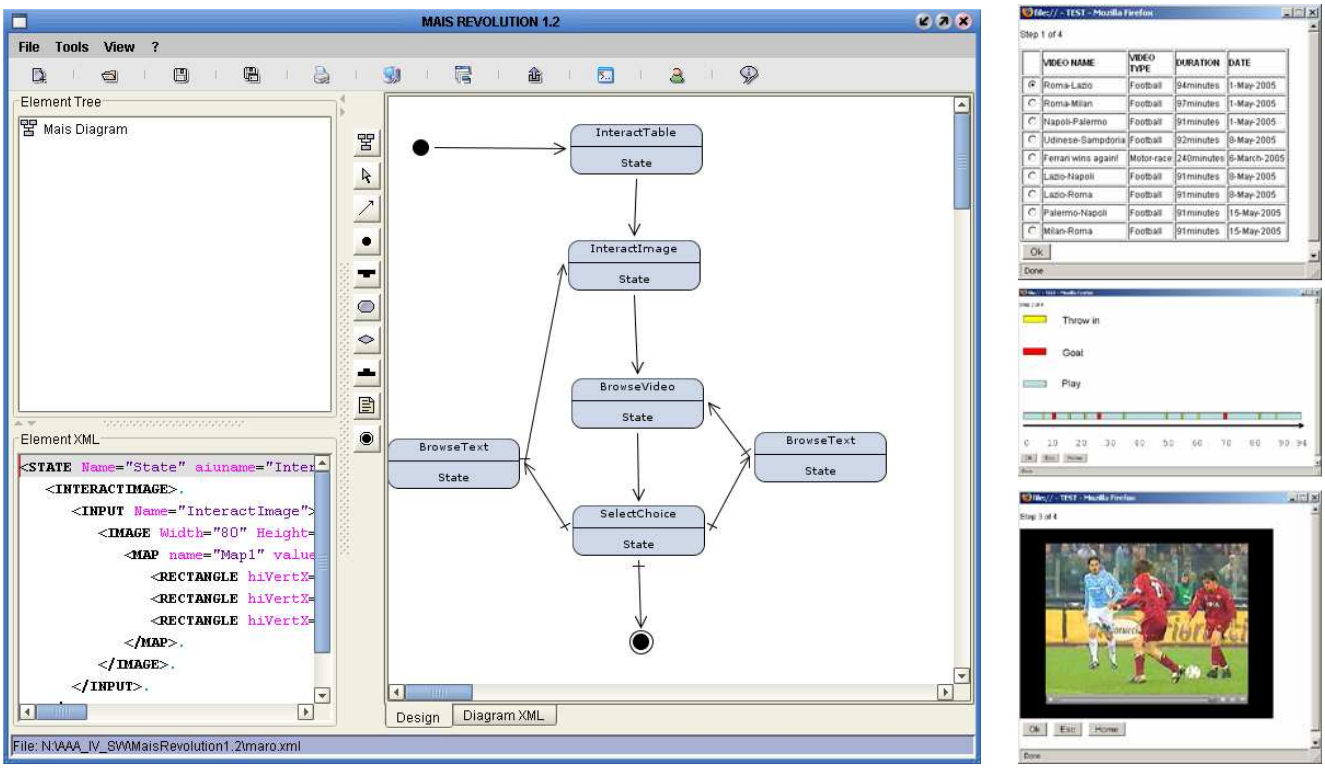

Fig. 1. *(a)Modeling a service for selectively browsing a video (b)Implementations of the IntractTable, InteractImage, BrowseVideo* `AIUs`

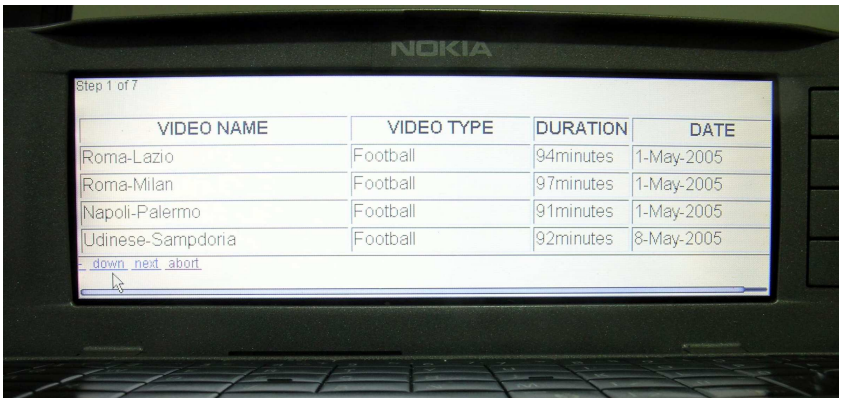

Fig. 2. *Implementing the InteractionTable* `AIU` *on a medium size handheld*

devices metrics; moreover we are developing a first set of usability studies, to validate our approach.

VII. Acknowledgments

Work supported by the MIUR-FIRB project "MAIS" (Multichannel adaptive Information Systems, and by DELOS NoE.

References

[1] S. Ceri, P. Fraternali, and A. Bongio. Web modeling language (webml): a modeling language for designing web sites. In *Proceedings of the 9th international World Wide Web conference on Computer networks*, pages 137–157. North-Holland Publishing Co., 2000.

[2] J. Eisenstein, J. Vanderdonckt, and A. Puerta. Applying model-based techniques to the development of uis for mobile computers. In *Proceedings of the 6th international conference on Intelligent user interfaces*, pages 69–76. ACM Press, 2001.

[3] P. Fraternali. Tools and approaches for developing data-intensive web applications: a survey. *ACM Comput. Surv.*, 31(3):227–263, 1999.

[4] F. Garzotto, P. Paolini, and D. Schwabe. HDM - a model-based approach to hypertext application design. *ACM Trans. Inf. Syst.*, 11(1):1–26, 1993.

[5] T. Isakowitz, E. A. Stohr, and P. Balasubramanian. RMM: a methodology for structured hypermedia design. *Commun. ACM*, 38(8):34–44, 1995.

[6] G. Mori, F. Paternò, and C. Santoro. Tool support for designing nomadic applications. In *Proceedings of the 2003 international conference on Intelligent user interfaces*, pages 141–148. ACM Press, 2003.

[7] A. Puerta and J. Eisenstein. Towards a general computational framework for model-based interface development systems. In *Proceedings of the 4th international conference on Intelligent user interfaces*, pages 171–178. ACM Press, 1999.

[8] D. Thevenin and J. Coutaz. Plasticity of user interfaces: Framework and research agenda. In *Proceedings of Interact'99*.

[9] M. van Welie and B. de Groot. Consistent multi-device design using device categories. In *Proceedings of the 4th International Symposium on Mobile Human-Computer Interaction*, pages 315–318. Springer-Verlag, 2002.